**VŠB – Technická univerzita Ostrava**
**Fakulta elektrotechniky a informatiky**

# DIPLOMOVÁ PRÁCE

**2017** **Bc. Adam Uhlíř**

**VŠB – Technická univerzita Ostrava**
**Fakulta elektrotechniky a informatiky**
**Katedra kybernetiky**
**a biomedicínského inženýrství**

# Multimediální a diagnostická jednotka pro osobní automobily

**2017** **Bc. Adam Uhlíř**

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

# Zadání diplomové práce

| | |
|---|---|
| Student: | **Bc. Adam Uhlíř** |
| Studijní program: | N2649 Elektrotechnika |
| Studijní obor: | 2612T041 Řídicí a informační systémy |
| Téma | Multimediální a diagnostická jednotka pro osobní automobily |
| | Multimedia and Diagnostic Unit for Cars |
| Jazyk vypracování: | čeština |

Zásady pro vypracování:

Diplomová práce se zabývá analýzou, návrhem a realizací multimediální a diagnostické jednotky pro osobní automobily. Výsledná aplikace je součástí multimediálního frameworku na platformě Linux s rozšířením pro online diagnostiku provozních veličin.

V souhrnu je práce charakterizována následujícími body:

1. Rešerše současného stavu multimediální techniky a diagnostiky v automobilech.
2. Analýza, návrh a realizace HW platformy pro jednotku.
3. Návrh a implementace SW modulu pro multimediální jednotku.
4. Návrh a implementace SW modulu pro diagnostickou jednotku.
5. Test jednotky.
6. Zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] BLOCH, Joshua. *Effective Java*. 2 ed., Addison-Wesley, 2008. ISBN 978-0321356680.
[2] ALBING, Carl a Michael SCHWARZ. *Java Application Development on Linux*. 1st ed. Prentice Hall, 2004. ISBN 978-0131436978.
[3] MCCORD, Keith. *Automotive Diagnostic Systems: Understanding OBD-I & OBD-II (S-A Design Workbench Series)*. CarTech, 2011. ISBN 978-1934709061.
[4] HORÁK, B., K. FRIEDRISCHKOVÁ, J. KAZÁRIK, J. NOWAKOVÁ a Z. SLANINA. *Elektromobilita II, učební text*. 1. vyd. Ostrava: VŠB-TU Ostrava, 2014. ISBN 978-80-248-3532-7. Dostupné z: http://netfei.vsb.cz/downloads/autorske_texty/Elektromobilita%20II.pdf.
[5] MONK, Simon. *Programming the Raspberry Pi, Second Edition: Getting Started with Python*. 2 ed., McGraw-Hill Education, 2015. ISBN 978-1259587405.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Zdeněk Slanina, Ph.D.**

Datum zadání: 01.09.2015

Datum odevzdání: 28.04.2017


doc. Ing. Jiří Koziorek, Ph.D.
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

## Prohlášení studenta

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 28. dubna 2017

……

Podpis

## Poděkování

Rád bych na tomto místě poděkoval panu Ing. Zdeňku Slaninovi, PhD. za podnětné rady a metodické vedení. Dále děkuji své rodině a blízkým za psychickou i materiální podporu.

## Abstrakt

Předmětem diplomové práce „Diagnostická a multimediální jednotka pro osobní automobily“ je návrh a realizace prototypu počítačové jednotky určené k montáži do osobních automobilů. Výsledná aplikace je součástí multimediálního frameworku na platformě Linux s rozšířením pro online sledování provozních veličin a diagnostiku provozních poruch.

## Klíčová slova

PC v automobilu, Raspberry Pi, Kodi, Arduino, FM rádio, OBDII, EOBD, Java, Akka, Actor Framework

## Abstract

The subject of thesis “Multimedia and diagnostic unit for cars” is the design and realization of prototype of computer unit designed for installation in passenger cars. The resulting application is part of a multimedia framework on the Linux platform with an extension for online monitoring of operating variables and diagnostics of trouble codes.

## Keywords

Car PC, Raspberry Pi, Kodi, Arduino, FM radio, OBDII, EOBD, Java, Akka, Actor Framework

# Obsah

## Seznam použitých symbolů a zkratek

| | |
|---|---|
| CAN | Controller Area Network |
| CNG | Compressed Natural Gas |
| CRC | Cyclic Redundancy Check |
| DC | Direct Current |
| DTC | Diagnostics Trouble Code |
| DVD | Digital Versataile Disc |
| ECU | Electronic Control Unit |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| EOBD | European Standard of OBD |
| FM | Frequency Modulation |
| GPIO | General Purpose Intput and Output |
| GPS | Global Positioning Systém |
| HDMI | High Definition Multimedia Interface |
| HID | Human Interface Device |
| HW | Hard Ware |
| IO | Input Output |
| ISO | International Organization for Standardzation |
| JDK | Java Development Kit |
| LCD | Liquid Crystal Display |
| LED | Light Emitting Diode |
| LPG | Liquified Petroleum Gas |
| MAC | Media Access Control |
| MCU | Micro Controller Unit |
| MIL | Malfunction Information Lamp |
| MMC | Multimedia centrum |
| OBD | On Board Diagnostics |

| | |
|---|---|
| OS | Operarating System |
| PC | Personal Computer |
| PID | Parameter ID |
| PWM | Pulse Width Modulation |
| SAE | Society of Automotive Engineers |
| SD | Secure Digital |
| SW | Soft Ware |
| TCP | Transfer Communication Protocol |
| UDP | User Datagram Protocol |
| USB | Universal Serial Bus |
| VIN | Vehicle Information Number |
| XML | eXtensible Markup Language |

# Seznam ilustrací

## Seznam tabulek

# 1 Úvod

Cílem práce je navrhnout a vytvořit prototyp multimediální jednotky s možností diagnostiky provozních veličin automobilu. Za platformu byl zvolen jednodeskový počítač Raspberry Pi 3 [8] s operačním systémem Raspbian [9], na kterém bude spuštěn multimediální framework Kodi [5] a virtuální stroj Javy.

Multimediální část by měla obsahovat FM tuner, GPS navigaci a multimediální centrum, které bude jedním z uživatelských vstupů. Diagnostika by měla umožňovat vyčítat a zobrazovat provozní veličiny automobilu, a také zobrazovat a dešifrovat chybové kódy ECU.

# 2 Rešerše dostupných řešení

Před započetím návrhu je výhodné provést nejprve rešerši již dostupných zařízení pro představu a inspiraci.

## 2.1 Multimédia v automobilech

Multimediální přehrávače jsou spojeny s provozem automobilu již hodně desítek let, počínaje jednoduchým rádiovým přijímačem, až po složité systémy, které mohou být v podstatě stejně výkonné jako průměrný laptop nebo chytrý telefon.

### 2.1.1 Kompletní vestavěné MMC

Největší přepych nabízí uživatelům kompletní multimediální centrum, nabízející většinu možností, které by mohl uživatel v automobilu uvítat. Jedná se například o navigaci GPS, přehrávač CD/DVD nebo WiFi připojení pro přístup k Internetu. Jednotlivá konstrukční řešení zobrazuje Obr. 1.

V podstatě lze místa montáže MMC rozdělit na montáž do palubní desky a mimo ni. Do palubní desky se nejčastěji používá MMC o výšce 2DIN, což odpovídá normalizované výšce modulů pro palubní desky v osobních automobilech (klasické autorádio má výšku 1DIN). Mimo palubní desku jsou pak nejoblíbenější místa buď přichycení ke stropu, nebo zabudování do opěrek sedadel. Méně používaná je pak montáž monitoru pomocí uchycovací konstrukce kamkoli ve vozidle.

### 2.1.2 Autorádia

Obr. 1 Zleva MMC do palubní desky (výška 2DIN), MMC zabudované do opěrek sedadel, MMC pro uchycení ke stropu

Jedná se asi o nejpoužívanější multimediální zařízení vhodné pro montáž do automobilu. Klasické rozhraní obsahuje CD mechaniku, LCD displej a řadu tlačítek. Softwarová konfigurace povětšinou podporuje většinu hudebních formátů a příjem rádiových vln v několika pásmech. U některých je navíc slot pro SD kartu či USB, příp. možnost spárovat se pomocí Bluetooth s telefonem.

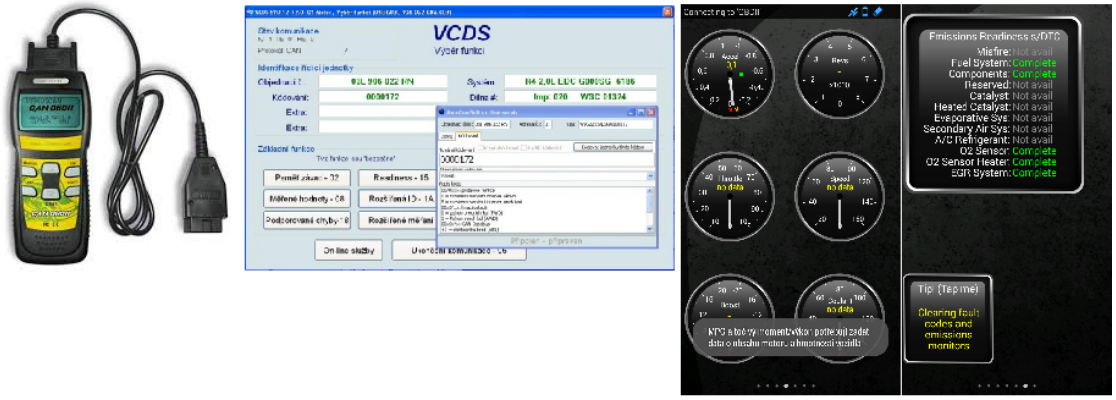
Většina autorádií má výšku 1DIN, některé ovšem mohou mít výšku i 2DIN, jak zobrazuje Obr. 2.


Obr. 2 Zleva Autorádio s výškou 1DIN, Autorádio s výškou 2DIN

## 2.2 Diagnostické systémy pro osobní automobily

Diagnostika automobilu uživatelem je v poslední době čím dál aktuálnější téma. Z toho důvodu vzniká spousta programů a zařízení, které umožní kontrolovat stav ECU i provádět změny, jako např. resetovat chyby, nastavovat parametry částí automobilu apod.

Diagnostické systémy lze rozdělit na dvě skupiny podle přístupného média. První skupinou jsou jednodušší systémy pro rychlou diagnostiku, které zobrazuje Obr. 3 vlevo. K detailní diagnostice potom slouží skupina druhá, kterou představuje široká škála programů. Tuto detailní analýzu vozidla lze provádět jak z PC, Obr. 3 uprostřed, tak z chytrého telefonu, Obr. 3 vpravo.




Obr. 3 Prostředky pro diagnostiku automobilů

# 3 Diagnostika osobních automobilů

Diagnostika osobních automobilů je čím dál rychleji se vyvíjející obor, především s nástupem vestavěných počítačů. Dnešní automobily obsahují až několik desítek jednotek, které kontrolují provozní funkce.

## 3.1 Historie

Historie diagnostiky se datuje od vynálezu prvních automobilů, ovšem diagnostikou v dnešním pojetí lze zamýšlet spíše diagnostiku pomocí počítačů, proto ani jiný typ diagnostiky nebude dále uváděn.

### 3.1.1 Historie před OBD-I

Diagnostika osobních automobilů pomocí počítačů začíná na počátku 80. let, kdy se začaly nasazovat první řídicí jednotky, které bylo potřeba nějakým způsobem spravovat. Tyto jednoduché počítače obstarávaly pouze řízení doby vstřiku pomocí údajů ze senzorů. Komunikační rozhraní u těchto jednotek tvořilo tzv. vyblikávání. To fungovalo tak, že někde na palubní desce byla umístěna LED dioda, která podle počtu bliknutí udávala kód, který v kombinaci s manuálem sdělil technikovi požadovanou informaci.

Primárním důvodem pro zavedení elektronického řízení vstřikovačů byla lepší kontrola emisí vozidla. Se zvětšujícími se nároky na emise vozidel bylo časem třeba přejít ke složitějším řídicím jednotkám. O emisním ústrojí automobilu pojednává kapitola 3.2.

### 3.1.2 OBD-I

Teprve koncem 80. a začátkem 90. let se začaly řídicí jednotky používat pro řízení většího počtu provozních veličin automobilů, což logicky vedlo i k většímu počtu řídicích jednotek umístěných po vozidle. Jelikož se elektronické řídicí jednotky vyskytovaly již téměř v každém nově vyrobeném vozidle a navíc nabízely mnohem větší funkcionalitu, bylo třeba nějakým způsobem sjednotit diagnostická rozhraní a protokoly. Prvním krokem k tomuto sjednocení se stalo rozhraní OBD-I, deklarované v roce 1989 v USA. Jedná se o konektor na palubní desce, který obsahuje piny sériové komunikace, která již umožňuje oboustrannou komunikaci. Každý výrobce však tento standard realizoval po svém, takže se ještě nedá mluvit o plném sjednocení diagnostiky automobilů.

### 3.1.3 OBD-II

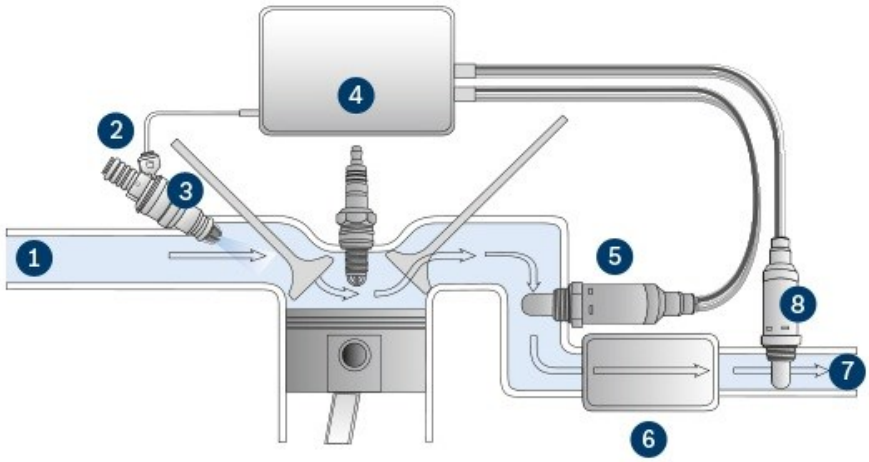
Významná změna v ohledu sjednocení diagnostických protokolů nastala až v roce 1996 v USA, kde byl definován standard OBD-II, který sjednocuje typ konektoru i komunikaci s jednotkou [1][4]. O tomto standardu hovoří více kapitola 3.3.1.

## 3.2 Popis vybraných zařízení souvisejících s emisemi vozidla

Emisní ústrojí vozidel je v dnešní době velmi diskutované téma, neboť je kladen mnohem větší důraz na čistotu ovzduší, než dříve. Z toho důvodu jsou níže podrobněji popsány dvě hlavní částí emisních rozvodů.

### 3.2.1 Lambda sonda

Nezbytným zařízením při řízení emisí spalování u zážehových (dnes už i vznětových) motorů je zařízení, které porovnává okolní vzduch s výfukovými plyny. Toto zařízení dává řídicí jednotce informaci, podle které může řídit bohatost vstřikovací směsi. Tímto zařízením je tzv. lambda sonda, která měří množství kyslíku na výstupu katalyzátoru, jak zobrazuje Obr. 4. Optimální bohatost směsi určuje tzv. stechiometrický poměr, který je označován jako λ=1. Tomuto poměru fakticky odpovídá zhruba 14,7 kg vzduchu na 1 kg benzínu. Technologie provedení senzoru se liší podle typu motoru i doby výroby automobilu [14].




Obr. 4 Schéma emisního potrubí motoru

Tabulka 1 Popis schéma emisního potrubí motoru

| Číslo zařízení | Název zařízení |
|---|---|
| 1 | Přívod vzduchu |
| 2 | Přívod paliva |
| 3 | Vstřikovač |
| 4 | Řídicí jednotka |
| 5 | Regulační sonda před katalyzátorem |
| 6 | Katalyzátor |
| 7 | Výfukové plyny |
| 8 | Diagnostická sonda za katalyzátorem (lambda sonda) |

### 3.2.2 Katalyzátor a kvalita spalování motoru

Katalyzátor výfukových plynů, někdy též zvaný katalytický konvertor, se používá pro snížení škodlivin, obsažených ve výfukových plynech motoru [15]. Je montován hned za motor, jak ukazuje Obr. 4.

Obsahuje kovovou či keramickou vložku, na které je nanesen některý ze vzácných kovů, např. rhodium nebo platina. Tyto vzácné kovy způsobují reakce, které mění zejména:

- kysličník uhelnatý na kysličník uhličitý
- uhlovodík na vodu
- kysličníky dusíku na dusík

Katalyzátory lze rozdělit podle použitého typu motoru na řízené a neřízené. Řízené katalyzátory se používají u zážehových motorů, neřízené u vznětových.

Dále podle použité vložky lze dělit na katalyzátory s keramickou vložkou a kovovou. Keramická je více odolná proti poškození a není vhodné ji používat i u motorů s alternativními pohony jako např. LPG nebo CNG, protože výfukové plyny těchto motorů jsou sušší a teplejší, a je tím pádem vyšší riziko prasknutí vložky. Oproti tomu kovová vložka vydrží mnohem větší fyzické i tepelné namáhání a je tedy vhodná i pro motory na LPG/CNG.

**Trojcestný katalyzátor**

Tento typ katalyzátoru umožňuje nejefektivnější čištění spalin, neboť obsahuje složky, odstraňující ze spalin všechny tři výše zmíněné látky najednou. Tuto funkcionalitu jim umožňuje kombinace keramických i kovových vložek, pokrytých vrstvami platiny, rhodia a india.

Jedinou nevýhodou těchto katalyzátorů je efektivní činnost až od určitých provozních teplot, proto se jim předřazuje další katalyzátor, který snižuje emisní zátěž už ve fázi zahřívání motoru.

Kvalita spalování v motoru závisí na mnoha faktorech. Ovšem většina těchto faktorů může být eliminována správným nastavením parametrů řídicí jednotky. Jedná se především o správné nastavení:

- bohatosti směsi (podle lambda sondy, příp. sondy mezi katalyzátorem a motorem)
- časování vstřikovacích ventilů
- časování výfukových ventilů
- časování zapálení stlačené směsi ve válci

Při správném nastavení výše uvedených parametrů motoru je palivo spalováno s nejvyšší možnou efektivitou, a proto také emise obsahují minimum škodlivin.

Na obsah škodlivin na výstupu mohou mít vliv i další faktory, které se odchytávají o poznání hůře. Jedná se například o nekvalitní palivo nebo palivo obsahující drobné nečistoty.

## 3.3 Standard OBD-II/EOBD

Standardy OBDII a EOBD jsou nástupci staršího OBD standardu, a jsou sjednocením pravidel pro počítačovou diagnostiku automobilů [1][4].

### 3.3.1 OBD-II

Standard OBD-II slouží k monitorování systémů pro regulaci emisí a klíčových komponent motoru. Princip činnosti tkví v provádění souvislých nebo periodických testů. V případě problému se rozsvítí příslušná kontrolka na palubní desce řidiče. Konektor je definován normou J1962. Má 16 pinů a zásuvka musí být umístěna na místě přístupném řidiči vozidla, nejdále však 60 cm od volantu. Provedení konektoru zobrazuje Obr. 5.

### 3.3.2 EOBD

EOBD je evropským ekvivalentem standardu OBD-II, od něhož se téměř v ničem neliší. Je definován pro vozidla kategorie M1, tedy vozidla mající méně než 8 míst k sezení a váhou nepřekračují 2500 kg. EOBD také disponuje normou J1962, definující stejný konektor jako u OBD-II.

Tabulka 2 Tabulka pinů diagnostického konektoru OBDII/EOBD

| Číslo pinu | Význam pinu |
|---|---|
| **1** | GM SAE J2411 (1-vodičový CAN s malou rychlostí) |
| **2** | J1850 PWM Bus+ nebo J1850 VPW Bus |
| **3** | Chrysler CCD+ (není OBD) |
| **4** | Kostra vozidla |
| **5** | Komunikační kostra |
| **6** | CAN-Bus High (J2284) |
| **7** | Komunikační linka K-line (ISO 9141-2) |
| **8** | Nespecifikováno |
| **9** | Nespecifikováno |
| **10** | J1850 PWM Bus- |
| **11** | Chrysler CCD- (není OBD) |
| **12** | Nespecifikováno |
| **13** | Nespecifikováno |
| **14** | CAN-Bus Low (J2284) |
| **15** | Inicializační linka K-line nebo 2. K-line (ISO 9141-2) |
| **16** | Palubní napětí +12V |

OBD-II definuje:

- diagnostický konektor
- komunikační protokol
- kódy pro jednotnou komunikaci s jednotkou

OBD-II dále předepisuje permanentní kontrolu:

- lambda-sondy
- katalyzátoru a spalování
- systémy sekundárního vzduchu, odvzdušnění nádrže a recirkulace spalin

## 3.4 Popis jednotlivých protokolů OBD-II/EOBD

Standard OBDII definuje 5 druhů komunikačních protokolů. Každé vozidlo většinou podporuje pouze jeden z nich [1][4].

### 3.4.1 SAE J1850

Tato norma definuje dva typy fyzické vrstvy:

- **VPW**

Jedná se o 1vodičovou komunikaci využívající variabilní šířku pulzu. Přenosové rychlosti mohou být buď 10.4 kbps nebo 41,6 kbps. Délka vedení max. 40 m a počet uzlů max. 32.

Tento standard navrhla společnost General Motors. Délka slova je 12 bitů včetně CRC. Protokol využívá metodu CSMA/NRP.

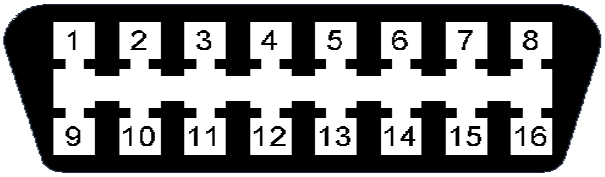
- **PWM**

Jedná se o 2vodičovou diferenciální komunikaci využívající pulzně-šířkovou modulaci. Přenosová rychlost je 41,6 kbps, délka vedení max. 40 m a počet uzlů max. 32.

Tento standard navrhla společnost General Motors pro vozy Ford. Délka slova je 12 bitů včetně CRC. Protokol využívá metodu CSMA/NDA.

### 3.4.2 ISO 9141-2

Jedná se o tzv. K-line komunikaci, která využívá asynchronní sériový přenos s rychlostí 10,4 kbps. Protokol je podporován převážně vozy značky Chrysler, evropskými a asijskými vozy. Jedná se o určitou obdobu RS232, ovšem s jinými napěťovými úrovněmi. Používané pakety mají délku 12 bitů a komunikace probíhá pomocí rozhraní UART.



Obr. 5 Diagnostický konektor OBD-II/EOBD

### 3.4.3 ISO 14230 KWP2000

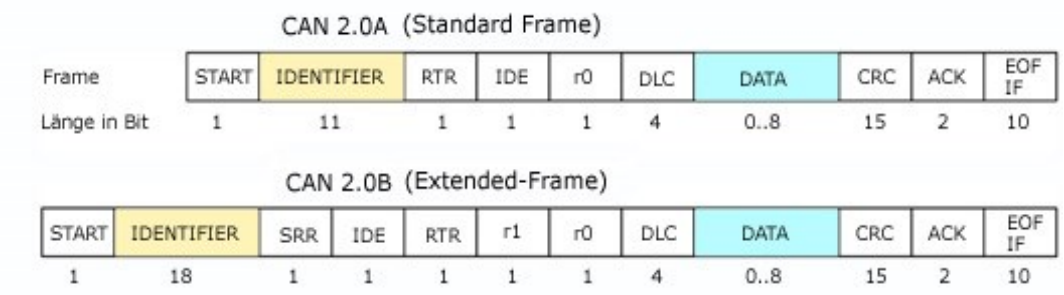
ISO 14230 je také známý jako tzv. Keyword Protocol 2000. V podstatě se jedná o obdobu protokolu ISO 9141-2 s několik rozdíly. Tyto rozdíly tvoří především větší délka paketů, které mohou být dlouhé až 255 bytů, a také větší rozpětí přenosových rychlostí od 1,2 kbps až do 10,4 kbps.

### 3.4.4 ISO 15765 CAN

CAN je sériová datová sběrnice, navržená v roce 1986 firmou Bosch. Cílem bylo vytvořit sběrnici, která povede k úspoře kabeláže a zároveň dokáže uspokojivě zabezpečit přenos informací.

Sběrnice může být tvořena jedním (single wire CAN) nebo dvěma vodiči (low nebo high speed CAN). Většinou se však používá 2-vodičové řešení, využívající kanály CAN_L a CAN_H. Logické úrovně jsou tvořeny napěťovým rozdílem mezi těmito dvěma kanály, konkrétní hodnoty však nejsou definovány. Rychlost sběrnice může být až 1 MB/s při délce vedení 40 m (high speed CAN).

Komunikace probíhá pomocí zpráv, které lze rozdělit podle normy pro CAN2.0A a CAN2.0B. Rozdíl tvoří především délka identifikátoru, který u provedení CAN2.0B dovolí adresovat větší počet zařízení. Tato norma se používá i u malého počtu zařízení na sběrnici a nevyužité bity se používají pro přenos bitových dat kvůli úspoře provozu na síti. Podobu rámce zobrazuje Obr. 6.




Obr. 6 Rámec CAN2.0

## 3.5 OBDII diagnostika

Automobilovou diagnostiku lze rozdělit v podstatě do třech částí:

- První část tvoří diagnostika podle mezinárodních standardů ISO a týká se pouze moderních automobilů. Jedná se o univerzální nástroj umožňující komunikovat maximálně s 8 ECU jednotkami najednou a zjišťovat základní poruchy.
- Druhou tvoří diagnostika profesionální umožňující hloubkovou analýzu všech jednotek v automobilu. Tuto diagnostiku používají především autorizované servisy, nebo ji umožňují speciální diagnostické přístroje určené přímo pro daný typ automobilu. Tyto přístroje se ovšem nedají sehnat bez autorizace výrobce.
- Třetí část tvoří tzv. chiptuning, který představuje úplnou kontrolu nad daty, a umožňuje výrobcem nepovolené modifikace nastavení jednotek.

### 3.5.1 OBDII/EOBD režimy

Jak je uvedeno v J1979 nebo v ISO 15031-5 podporuje OBDII 10 režimů [1][4]:

Tabulka 3 Tabulka OBDII režimů

| Režim | Popis |
|---|---|
| **1** | Monitorování aktuálních hodnot |
| **2** | Používá se pro „freeze frame data“, tedy data rámce, při kterém nastala porucha |
| **3** | Zobrazí uložené diagnostické poruchové kódy |
| **4** | Smazání uložených poruchových kódů |
| **5** | Výsledky zkoušek a test lambda sond (nefunkční pro CAN) |
| **6** | Výsledky zkoušek a pro CAN test lambda sond |
| **7** | Zobrazí dočasné chybové kódy týkající se emisí za poslední jízdní cyklus |
| **8** | Řízení provozu palubního systému a konstrukční části |
| **9** | Získá informace o vozidle (VIN, CALIB_ID, CVN) |
| **A** | Chybové kódy uložené v EEPROM |

### 3.5.2 PID příkazy

Pro jednotlivé režimy jsou definovány tzv. PID (Parameter ID), které slouží ke čtení těch dat v jednotce, které dostává ze snímačů nebo které jsou výsledkem výpočtů [2].

Většina PID příkazů je definována normou J1979, ovšem některé jsou definovány samotnými výrobci, většinou pro nestandardní data. Více o PID příkazech pojednává kapitola 7.9.1.

### 3.5.3 Závady definované standardem OBDII/EOBD

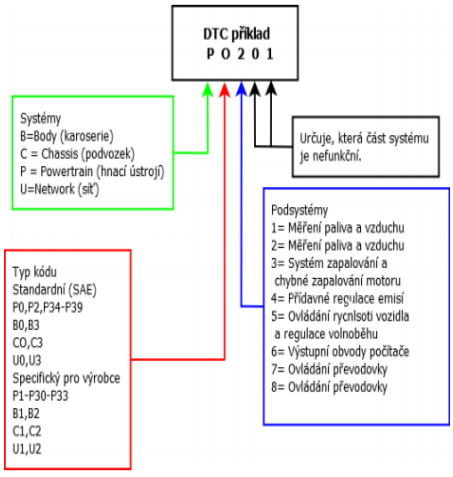
Standard definuje 4 typy závad označených od A po D, přičemž závady typu D přímo neovlivňují emisní systém [4]. Tabulka 4 zobrazuje popis jednotlivých typů závad.

Tabulka 4 Typy závad OBDII/EOBD

| Typ závady | Popis |
|---|---|
| **A** | Tato porucha představuje nejvážnější situace, které jsou indikovány rozsvícením kotrolky MIL a do paměti jsou uloženy provozní podmínky, při kterých k poruše došlo. V případě poruchy typu A se vždy jedná o poruchu související s emisním systémem. |
| **B** | Tento typ poruchy není tak vážný jako typ A. Jedná se o poruchu, která se objeví pouze v jednom po sobě jdoucích cyklů. Pokud je závada signalizována kontrolkou MIL, uloží se provozní podmínky do paměti. Také u poruch typu B se vždy jedná o poruchu související s emisním systémem. |
| **C** | Porucha typu C má nižší závažnost než porucha typu B, a nemusí vždy souviset s emisním systémem. Pokud ovšem porucha typu C vede k rozsvícení kontrolky MIL, jsou opět uloženy provozní podmínky. tato porucha může mít na svědomí i rozsvícení jiného typu kontrolek. |
| **D** | Při tomto druhu poruchy nikdy nedojde k rozsvícení kontrolky MIL. Tato porucha se totiž nikdy nevyskytuje při poruchách emisního systému. |

### 3.5.4 Diagnostické kódy poruch DTC

Tyto kódy slouží k snadnějšímu odhalení oblasti poruchy a jsou uloženy přímo v ECU. Příklad kódu DTC demonstruje Obr. 7. Více o chybových kódech pojednává kapitola 7.9.2.




Obr. 7 Příklad DTC kódu

# 4 Návrh a realizace HW platformy

Návrh HW platformy představuje výběr všech potřebných HW komponent pro správnou funkci celého systému. Realizace pak spočívá v jejich montáži do cílového systému a jejich vzájemném propojení.

## 4.1 Vybrané komponenty

Po provedení krátké rešerše byly vybrány následující komponenty. Při jejich výběru byl brán ohled na rozměry a možnou rozšiřitelnost o další funkce.

### 4.1.1 Raspberry Pi 3

Raspberry Pi 3 je jednodeskový počítač obsahující výkonný procesor, což z něj, spolu s dobrým grafickým čipem, dělá dobrou volbu pro použití jako multimediální centrum [8].

Tabulka 5 Vlastnosti Raspberry Pi 3

| **Vlastnost** | **Hodnota** |
|---|---|
| Proccesor | Quad Core Broadcom BCM2837 64-bit ARMv8 (1.2 GHz) |
| Operační paměť | 1 GB LPDDR2 (sdílená s grafikou) |
| GPU | VideoCore IV 250 MHz (1080p30, H.264, MPEG-2, VC-1) |
| Analog video | Kompozitní (NTSC/PAL) video výstup je integrován do 4-pólového 3.5mm "sluchátkového" jacku |
| Digital video | HDMI |
| Pinová lišta | GPIO, I2C, SPI, UART |
| WiFi | BCM43143 |
| Úložiště | microSD |
| Rozměry | 85mm x 56mm |
| Napájení | MicroUSB |
| Výbava | 4× USB 2.0, HDMI, 100Mbit ethernet, 2×20pinový GPIO, CSI/DSI rozšíření pro kameru a dotykový displej |

Obr. 8 Raspberry Pi 3

### 4.1.2 Napájecí převodník pro Raspberry Pi 3

Napájení Raspberry Pi 3 je poměrně náročná záležitost, neboť při připojených periferiích může dosahovat spotřeby až 2,5 A. Vybraný převodník má odolné opláštění, protože bude umístěn mimo skříňku, aby zbytečně nezvyšoval teplotu uvnitř skříňky. Napětí v automobilu je 12 V.

Tabulka 6 Vlastnosti převodníku napětí

| **Vlastnost** | **Hodnota** |
|---|---|
| Typ | DC/DC |
| Vstupní rozsah | 8 V – 30 V |
| Výstupní napětí | 5 V |
| Výstupní proud | 5 A |

Obr. 9 Převodník 12V/5V 5A

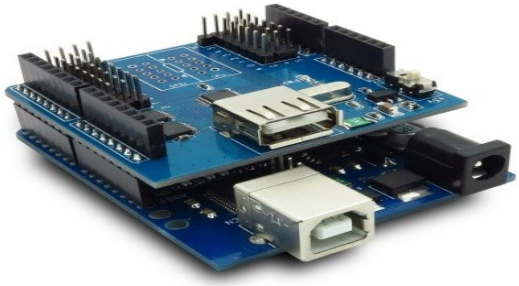
### 4.1.3 Ethernet shield pro Arduino Uno

Shieldy pro Arduino mají stejné pinové provedení, což umožňuje skládat tyto moduly na sebe a všechny společně přímo propojit se základní deskou. Ethernet shield umožňuje propojení s Raspberry Pi.

Obr. 10 Arduino Uno s Ethernet shieldem

### 4.1.4 USBHost shield pro Arduino Uno

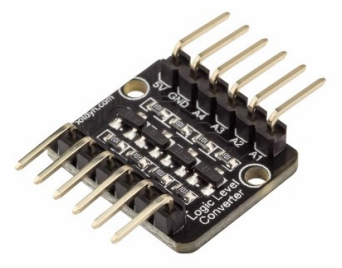
Tento typ shieldu umožňuje připojení HID zařízení k Arduinu. Je použit pro připojení tlačítkové lišty, která ovládá relé a rádio.



Obr. 11 Arduino s USB shieldem

### 4.1.5 Převodník logických úrovní

Použití tohoto druhu převodníku je nutné kvůli napěťovým úrovním datových sběrnic Arduina, které pracují s 5V logikou. Naproti tomu procesor Si4703, použitý v FM modulu akceptuje pouze 3,3V logiku.




Obr. 12 Převodník logických úrovní

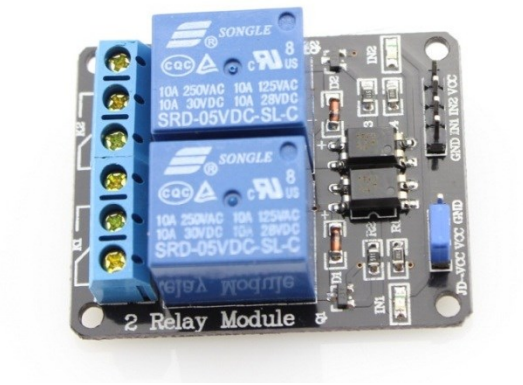
### 4.1.6 Zesilovač LM386

Pro připojení zvukových výstupů FM modulu a Arduina k reproduktorům je nutné použít zesilovač. Zesilovač LM386 splňuje požadavky pro připojení k 5W reproduktorům v automobilu. Obsahuje potenciometr pro nastavení hlasitosti, který je nastaven na 80% a hlasitost je dále ovládána softwarově.

Obr. 13
Zesilovač LM386

### 4.1.7 Reléový modul

Reléový modul má za úkol spínat napětí mezi převodníkem a Raspberry Pi. Je ovládán z Arduina po stisku tlačítka na předním panelu.

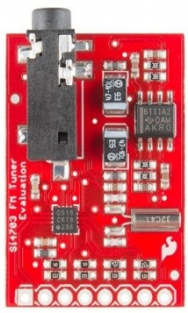



Obr. 14 Reléový modul

### 4.1.8 Modul s FM tunerem Si4703 a zesilovačem TPA6111A2

Obvody Si4702/03 společnosti SiliconLabs tvoří zcela integrované řešení digitálního FM přijímače. Vycházejí z architektury Si4700/01, se kterou jsou plně kompatibilní. Díky této návaznosti na již osvědčený projekt mohou nabídnout výborné vlastnosti:

- Šířka podporovaného pásma 32 MHz (76 MHz – 108 Mhz)
- Automatické řízení frekvence AFC
- Automatické řízení zisku AGC
- Zesilovač TPA6111A2 o výkonu 150 mW
- Podpora automatického hledání stanic
- Měření síly signálu
- Adaptivní potlačení šumu
- Výborná odolnost proti zahlcení
- Napájecí napětí 3,3 V



Obr. 15 FM modul Si4703

### 4.1.9 GPS modul Columbus V800

GPS lokátory mají často problém najít signál. Je proto dobré mít dostatečnou rezervu v možném umístění GPS modulu v automobilu. Tento požadavek výborně splňuje modul Columbus V800, který má velmi odolné provedení s USB konektorem.


Obr. 16 GPS modul Columbus V800

### 4.1.10 Diagnostický modul ELM327

ELM327 je MCU od firmy ELM Electronics pro komunikaci s řídicími jednotkami moderních aut. Většinou má funkci tzv. překladače, kdy hraje roli prostředníka v komunikaci s řídicí jednotkou. Výrobce nabízí bezdrátovou verzi s Bluetooth a verzi s USB. Po provedení testů byla vybrána verze s USB kvůli výrazně nižší latenci.

**Seznam podporovaných funkcí:**

- Čtení a výmaz chybových kódů motoru
- Čtení live dat
- Diagnostika lambda sondy
- Podpora velké řady protokolů:
    - ISO 15765-4 CAN A i B, 250 kbaud i 500 kbaud
    - ISO 14230-4 KWP
    - ISO 9141-2
    - J1850 VPW i PWM
- Podporován většinou profesionálních diagnostických SW

Obr. 17 Modul ELM327 s USB

### 4.1.11 Dotykový displej Samsung

Při výběru displeje byl kladen velký důraz na kvalitu obrazu, vysokou svítivost a přívětivé pracovní podmínky pro provoz v automobilu, kde mohou v zimě teploty dosahovat i -20°C.

Tabulka 7 Parametry displeje

| Vlastnost | Hodnota |
|---|---|
| **Model** | LMS700KF07-004 |
| **Velikost displeje** | 7 palců |
| **Rozlišení** | 1920x1080 |
| **Typ displeje** | Rezistivní dotykový |
| **Barevná hloubka** | 24 bitů |
| **Svítivost** | 300 cd/m2 |
| **Spotřeba** | 1.2 – 2 W |
| **Pracovní teplota** | -20°C až 60°C |
| **Úhly pohledu** | L70° R70° U50° D60 |

Obr. 18 Displej Samsung

### 4.1.12 Tlačítková lišta pro display Samsung

Na Obr. 18 lze vidět dvě tlačítkové lišty po stranách. Lišta s dvěma tlačítky je součást ovladače displeje a slouží k ovládání displeje. Druhá lišta je přídavná a její a signály lze použít pro vlastní účely pomocí USB rozhraní. V tomto případě bude sloužit jako kontrola rádia a vypínač pro Raspberry Pi.

Obr. 19 Uživatelská tlačítková lišta

## 4.2 Návrh zapojení

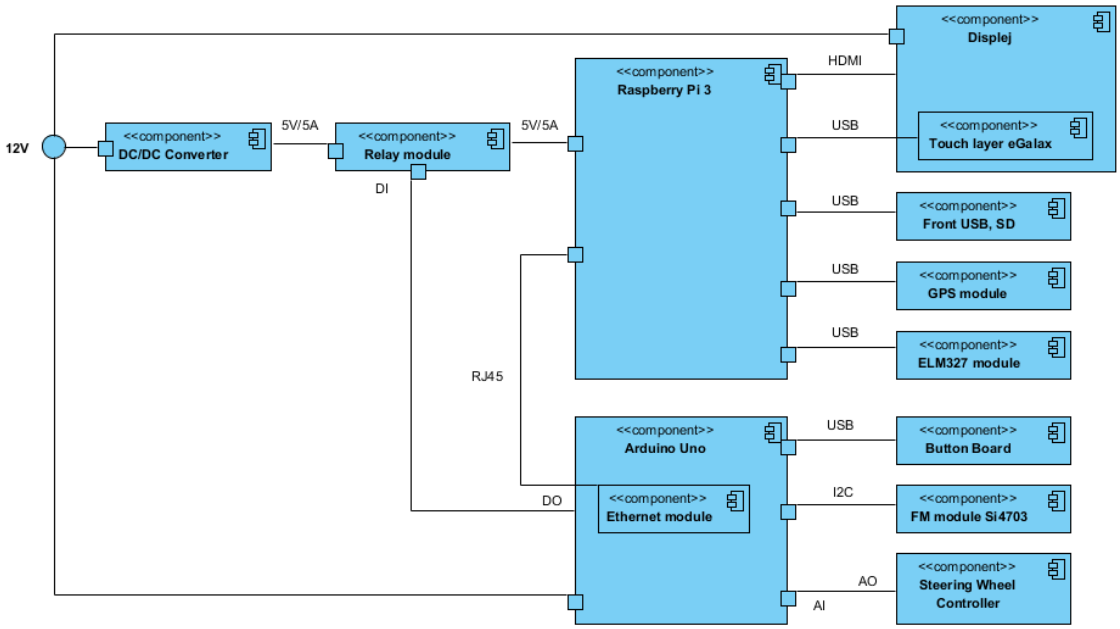
Spojení komponent je realizováno pomocí dostupných multimediálních konektorů i protokolů. Jejich úplný seznam včetně ostatních použitých doplňků uvádí Příloha A.

### 4.2.1 Napájení

Napětí v automobilu je 12V, což je přímo použitelné pro napájení Arduina a displeje. Pro napájení Raspberry Pi je však nutno použít napětí 5 V. Proto je do obvodu vložen stejnosměrný měnič na 5V/5A, který je navíc spínán pomocí relé, ovládaném digitálním výstupem z Arduina.

### 4.2.2 Schéma zapojení

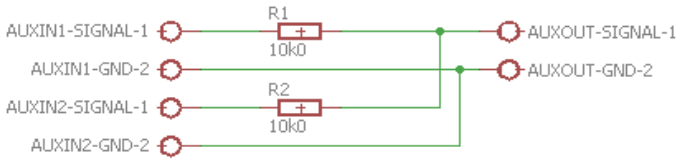
Propojení jednotlivých částí na Obr. 20 napovídá, v jakém pořadí se tyto části budou spouštět. Jako první se spustí Arduino, které sleduje stav tlačítek čelního panelu a na základě stisknutí tlačítka sepne napájení pro Raspberry Pi. Displej má samostatné ovládání, takže je možné ho připojit přímo na 12 V.




Obr. 20 Schéma návrhu zapojení HW

### 4.2.3 Spojení zvukových kanálů

V práci se vyskytují dvě zařízení se zvukovými výstupy, které je nutno spojit ještě před zapojením do zesilovače podle schématu na Obr. 21.




Obr. 21 Schéma spojení zvukových kanálů

## 4.3 Realizace platformy

K realizaci HW platformy byla použita skříňka připravená pro montáž do palubní desky. Komponenty jsou v ní uspořádané nad sebou, aby bylo optimálně využito volného prostoru, s přihlédnutím k tomu, že velkou část místa zabere kabeláž.

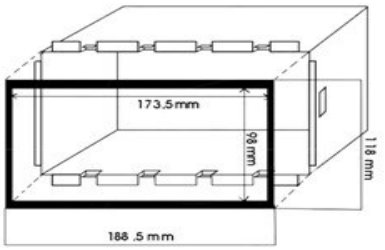
### 4.3.1 Uzemnění komponent

Komponenty jsou připevněny kovovými distančními sloupky k desce, která má vespod rozlitou měď. Taktéž konstrukce skříňky je k této desce připevněna kovovými šrouby a celá deska je pak uzemněna ke vstupní zemi.

### 4.3.2 Rozmístění komponent

Komponenty jsou rozmístěny s ohledem na jejich tepelný výkon. Napěťový převodník je umístěn mimo skříňku, aby nezvyšoval vnitřní teplotu. Raspberry Pi je umístěné u větracího otvoru, jelikož se očekává zvýšený tepelný výkon v letních měsících.

Obr. 22 Realizace HW platformy




Obr. 23 Rozměry skříňky 2DIN

# 5 Analýza a návrh SW platformy

Analýzu SW platformy lze rozdělit na dvě části. První část představuje analýza operačního systému, o čemž hovoří kapitoly 5.1.1 až 5.1.3. Druhá část analýzy se zabývá výběrem jazyka pro SW spojení všech komponent, o čemž pojednávají kapitoly 5.1.5 a 5.1.6.

## 5.1 Analýza dostupných komponent

Dostupných komponent je poměrně hodně, po provedené analýze se ovšem ukázalo, že vhodných je poměrně málo. Nesplňují totiž požadavky pro všechny vytyčené cíle zároveň.

### 5.1.1 Raspbian

Raspbian [9] je OS optimalizovaný pro Raspberry Pi a je založený na distribuci Debianu. Jako jediný představuje plnohodnotný operační systém, který umožňuje instalaci klasických linuxových programů nebo Javy [10].

### 5.1.2 Windows 10 IOT Core

Verze Windows pro Raspberry Pi 3 představuje pouze rozhraní pro vývoj aplikací v C#.NET pro Raspberry Pi. Není to tedy plnohodnotný OS a jeho možnosti jsou poměrně omezené.

### 5.1.3 LibreELEC

OS přímo postavený pro běh MMC Kodi (nástupce XBMC). Umožňuje přístup do příkazové řádky, ale s velmi omezenou množinou použitelných příkazů.

### 5.1.4 Kodi

Multimediální framework Kodi [5] nabízí širokou škálu funkcí ať už pro vývoj aplikací v Pythonu, nebo stahování již hotových aplikací, tzv. addonů, z různých internetových repozitářů. Tyto repozitáře jsou buď oficiální, nebo je mohou provozovat tzv. třetí strany, vyvíjející pro Kodi.

### 5.1.5 C#.NET

Vývoj aplikace v C#.NET by byl možný pouze při použití systému Windows 10 IoT Core, což je poměrně nevýhodné. Pro Linux lze použít Mono#, což je obdoba C#.NET, není to však plnohodnotná náhrada.

### 5.1.6 Java

Vývoj v Javě má nespornou výhodu z pohledu přenositelnosti kódu. Na Raspberry Pi je plně funkční a proto je lepší volbou než Mono#, což je varianta C#.NET pro Linux. Je zde samozřejmě i celá řada knihoven určených přímo pro Raspberry Pi, např. pro práci s I2C, SPI, AI nebo GPIO. Žádná z nich ale není v tomto projektu použita.

## 5.2 Výběr vhodné SW platformy

Po provedení testů jednotlivých platforem byl vybrán Raspbian, na kterém bude spuštěn virutální stroj Javy spolu s Kodi.

# 6 Návrh řešení pro Raspberry Pi

Návrh řešení pro Raspberry Pi představuje návrh aplikace v Javě, a addonu pro Kodi. Zmíněná aplikace v Javě tvoří zaprvé komunikační most mezi Kodi a Arduinem, zadruhé je jádrem celé diagnostické části.

## 6.1 Architektura

Systém se skládá z řídicí aplikace vytvořené v Jazyce Java a z Kodi addonů, sloužících jako uživatelský vstup, vytvořených v jazyce Python. K systému se bude připojovat také vzdálený systém běžící na Arduinu. Všechny aplikace komunikují s řídicí aplikací po vzoru klient-server pomocí TCP protokolu. Vnitřní architektura řídicího systému je vytvořena také po vzoru klient-server. Více v kapitole 7.2.

## 6.2 Model

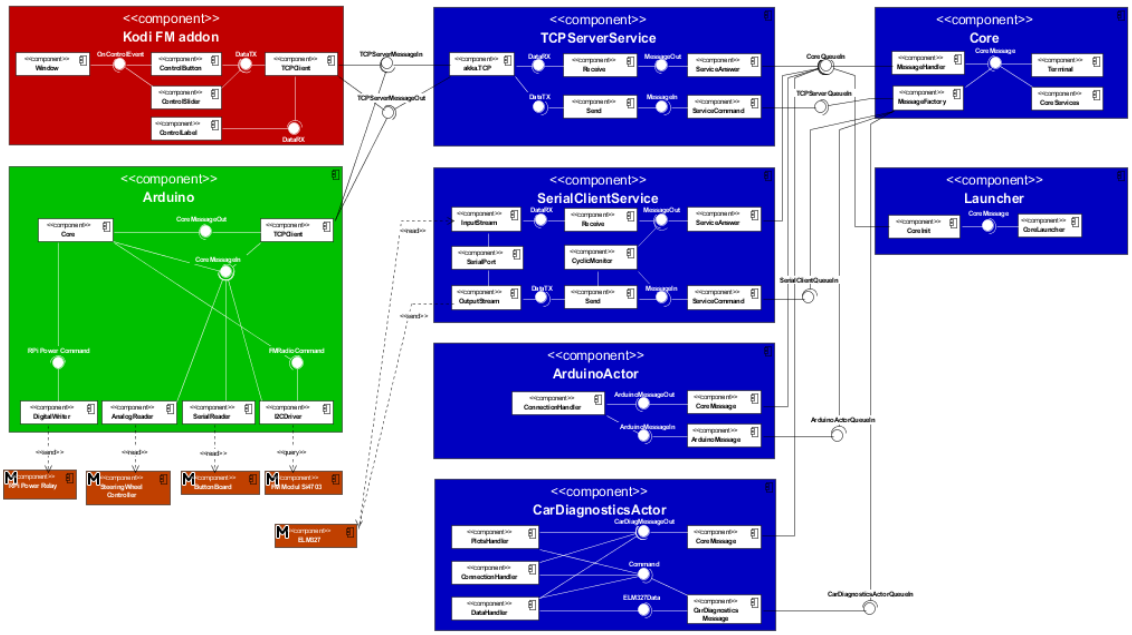
Celkový model na Obr. 24 představuje veškerý SW vytvořený pro potřeby této práce. Jednotlivé části jsou barevně zvýrazněny.

Modrou barvou je zvýrazněna hlavní řídicí aplikace, která je základním kamenem celého konceptu. Má na starosti zpracování uživatelských vstupů a komunikaci mezi jednotlivými částmi. Součástí řídicí aplikace je i diagnostika automobilů. Více v kapitole 7.2.

Červenou je zvýrazněn Kodi addon, který slouží jako hlavní uživatelský vstup a výstup pro FM rádio. Více v kapitole 7.10.2.

Zelenou barvou je zvýrazněno Arduino, které poskytuje funkcionalitu FM rádia a také slouží jako spouštěč a vypínač Raspberry Pi. Více v kapitole 7.6.

Komponenty s oranžovou barvou představují HW periferie systému, o kterých pojednává kapitola 4.1.




Obr. 24 Celkový model aplikace

## 6.3 Actor Framework

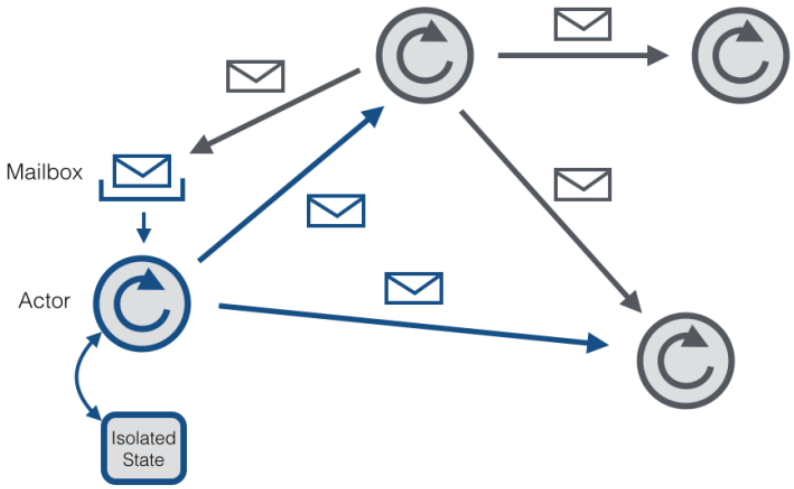
Řídicí aplikace je vytvořena v tzv. „Actor Frameworku“, což je moderní nástroj pro snadné vytváření vícevláknových, příp. distribuovaných aplikací.

Model aktorů je založen na topologicky uspořádané soustavě uzlů, kde každý uzel (aktor) má svou schránku na přijímání zpráv (mailbox) a vlastní stav, ke kterému má výlučný přístup. Aktoři pak spolu v rámci konkrétní aplikace komunikují výhradně zasíláním zpráv, na základě těchto zpráv mění svůj stav a odesílají další zprávy, jak zobrazuje Obr. 25.

Zprávy se řadí aktorům-adresátům do mailboxů a ti je zpracovávají, typicky v pořadí, v jakém přišly. V jednom časovém okamžiku zpracovává daný aktor maximálně jednu zprávu (ne více zpráv najednou). Tento přístup má jeden velmi důležitý důsledek: V jednom časovém okamžiku je potřeba jen jedno vlákno, které může měnit interní stav aktora. Stavové informace aktora tak není potřeba synchronizovat vůči přístupu více vláken najednou, protože taková situace nikdy nenastane. Tento způsob komunikace zároveň řeší problém kritické sekce, kdy pro přístup k jednomu sdílenému prostředku lze použít jednoho aktora.

### 6.3.1 Akka

Akka [11] je Actor frameworkem pro Javu. Jednotliví aktoři žijí v rámci jednoho systému. Aplikace se pak může skládat z více takovýchto systémů. V rámci systému aktorů lze využít některé služby. Některé z nich ani nejsou ve své nativní podobě použitelné kvůli tomu, že správu nad vlákny provádí Akka. Příkladem může být např. TCP připojení. Tento typ spojení (současně s UDP) poskytuje Akka prostřednictvím reference na aktora, který vykoná všechny potřebné úkony, resp. mu je třeba říct, aby je udělal prostřednictvím zpráv.




Obr. 25 Actor framework topologie

# 7 Realizace SW pro Raspberry Pi

Realizaci SW představuje především návrh multivláknového systému, který obslouží diagnostiku a umožní komunikovat jednotlivým částem systému. Celý projekt byl vyvíjen buď v IDE Eclipse s rozšířením WindowBuilder, nebo s použitím NotePad++.

## 7.1 Instalace nezbytného SW

Pro správnou funkci systému je nutné provést před započetím práce instalaci nezbytných částí systému, především instalaci Javy, Kodi a Navitu.

### 7.1.1 Update a upgrade OS

Pro update a upgrade je třeba zadat příkaz:

```
sudo apt-get update && sudo apt-get upgrade
```

Rozdíl mezi updatem a upgradem je následující:

- `apt-get update` – Stahují se nové soubory, nesoucí informace o balících a verzích
- `apt-get upgrade` – Aktualizují se všechny nainstalované balíky na nejnovější verzi

### 7.1.2 Java

Instalace JDK se provede příkazem

```
sudo apt-get install oracle-java8-jdk
```

### 7.1.3 Kodi

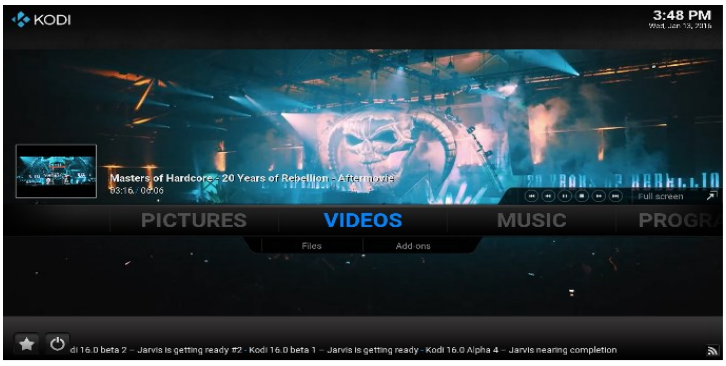
Pro instalaci Kodi stačí zadat jednoduchý příkaz

```
sudo apt-get install kodi
```

Důrazně se ovšem doporučuje provést předtím update a upgrade systému.

### 7.1.4 Xrdp

Xrdp je služba vzdálené plochy, která funguje jak přes kabel, tak pomocí WiFi. Použití tohoto nástroje významně urychluje vývoj.




Obr. 26 MMC Kodi

### 7.1.5 Navit

Navit je navigační systém vhodný pro většinu známých OS. Podporuje všechny známé funkce navigačních SW např. vyhledávání tras nebo hlasovou navigaci. V současné době podporuje 49 jazyků. Instalace navigačního systému Navit představuje stažení balíčku z repozitáře

```
svn://svn.code.sf.net/p/navit/code/trunk/navit/
```

a jeho kompilaci na cílovém systému. Této kompilaci ovšem musí předcházet instalace řady knihoven nezbytných pro správné fungování systému. Příkazy pro správnou instalaci těchto knihoven zobrazuje v chronologickém pořadí Tabulka 8.

Tabulka 8 Posloupnost příkazů instalace Navit

| **Popis příkazu** | **Příkaz** |
|---|---|
| Prvním krokem je vždy update operačního systému | `sudo apt-get update` |
| Instalace nezbytných knihoven pro pokračování v instalaci | `sudo apt-get install subversion imagemagick libdbus-1-dev libdbus-glib-1-dev libfontconfig1-dev libfreetype6-dev libfribidi-dev libimlib2-dev librsvg2-bin libspeechd-dev libxml2-dev ttf-liberation libgtk2.0-dev` |
| Pro správnou kompilaci je třeba instalovat následující kompilátory | `sudo apt-get install gcc g++ cmake make zlib1g-dev libpng12-dev librsvg2-bin` |
| Instalace ovladačů grafického enginu SDL | `sudo apt-get install libsdl-image1.2-dev libdevil-dev libglc-dev freeglut3-dev libxmu-dev libfribidi-dev` |
| Instalace ovladačů grafického enginu OpenGL | `sudo apt-get install libglc-dev freeglut3-dev libgl1-mesa-dev libfreeimage-dev` |
| Instalace ovladačů grafického enginu QT | `sudo apt-get install libqt4-dev` |
| Instalace podpory programu gpsd | `sudo apt-get install libgps-dev` |
| Instalace programu espeak | `sudo apt-get install espeak` |
| Instalace podpory Garmin map (není nutné, již podporuje Openstreet) | `sudo apt-get install libgarmin-dev` |

Po instalaci knihoven a stažení programu z výše uvedeného repozitáře je již možné přistoupit k samotné kompilaci příkazem

```
make -j4
```

Přepínač –j4 je nutné uvést pouze v případě Raspberry Pi 2 a 3. Po dokončení kompilace je již možné spustit program Navit příkazem

```
./navit
```

## 7.2 Návrh řídicí části

Návrh řídicí části obnáší především návrh systému aktorů a zpráv mezi nimi. Po dokončení návrhu následuje implementace.

### 7.2.1 Návrh systému aktorů

Při návrhu systému jsem se řídil osobní zkušeností s Actor Frameworkem a použil centralizovaný model. Ten představuje komunikaci mezi podřízenými uzly a jádrem. Jádro navíc zobrazuje terminál, kde lze nastavovat a zobrazovat prvky jednotlivých částí systému nebo prohlédnout a uložit log.

Jednotliví aktoři se dělí do dvou skupin podle typu:

- **Komunikační**
  Mají na starosti výměnu dat. Jsou vytvořeny univerzálně a lze jich spustit více. Typicky se spouští jeden aktor pro jeden port.

- **Modelové**
  Obsahují logiku jednotlivých funkcí systému. Mohou využít služeb jádra, typicky TCP a sériového spojení se vzdálenými komponentami.

## 7.3 Core

Funkcemi jádra jsou správa služeb a jejich distribuce jednotlivým modelům. Jádro je prvním a zároveň posledním spuštěným aktorem.

### 7.3.1 Služby

Modeloví aktoři při svém startu požádají jádro o vytvoření dané služby, která poté patří k danému aktoru. Jednotlivé služby jsou startovány jako aktoři, kde každý jeden představuje vlákno s vytvořeným spojením na daném portu.

Služby poskytované jádrem jsou:

- TCP server, více v kapitole 7.5
- Serial klient, více v kapitole 7.4

### 7.3.2 Zprávy pro jádro

Zprávy pro jádro představují skupinu zpráv, která má přímý vliv na chování systému:

**InitCore**

Po přijetí zprávy proběhne prvotní inicializace jádra, načtení konfigurací ze souborů a start modelových aktorů

**KillApp**

Po přijetí zprávy jádro odešle tzv. PoisonPill zprávu všem aktorům a čeká na jejich ukončení. Poté ji pošle samo sobě a aplikace skončí.

**ReturnSettings**

Požadavek terminálu na odeslání nastavení daného aktora.

**UpdateSettings**

Odeslání nastavení do terminálu.

**ServiceSetup**

Kontejner pro zapouzdření nastavení jednotlivých aktorů pro zobrazení v terminálu.

**UseService**

Touto zprávou žádají modeloví aktoři jádro o přidělení služby. Zpravidla tak, že v této zprávě pošlou zprávu ServiceCommand nastavenou na Init dané služby.

**ServiceCommand**

Zpráva pro použití služby.

**ServiceAnswer**

Zpráva od služby.

**TCPServiceCommand**

Zpráva pro nastavení nebo použití TCP serveru.

**TCPServiceAnswer**

Odpověď od TCP serveru.

**SerialServiceCommand**

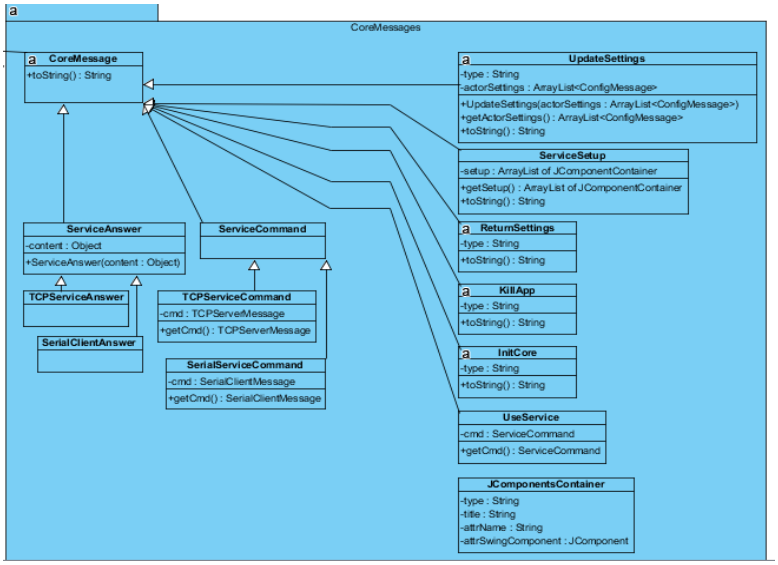
Zpráva pro nastavení nebo použití Serial klienta

**SerialServiceAnswer**

Odpověď od Serial klienta.

**JComponentContainer**

Kontejner pro přenos JComponent do terminálu.




Obr. 27 Zprávy pro jádro

## 7.4 SerialClient

SerialClient je univerzální předpis vlákna pro komunikaci sériovým portem jako klient. Tohoto typu komunikace je využito při komunikaci s rozhraním ELM327.

### 7.4.1 Služby

Aktor nabízí službu CyclicMonitor, díky které je možné cyklicky provádět funkce aktora. Služba funguje tak, že vlastník této služby registruje v CyclicMonitoru tzv. Variables, které jsou po daném časovém intervalu automaticky odeslány prostřednictvím zprávy Send. Těchto CyclicMonitor služeb je možné registrovat neomezené množství, pokud má každá jiný interval obnovení.

Cyklické opakování je zajištěno použitím plánovače, který je součástí frameworku Akka. Tento plánovač odešle jednou za časový interval danou zprávu danému aktoru.

### 7.4.2 Zprávy pro SerialClienta

SerialClient nabízí několik služeb, které mohou uživatelé služeb využívat.

**Init**

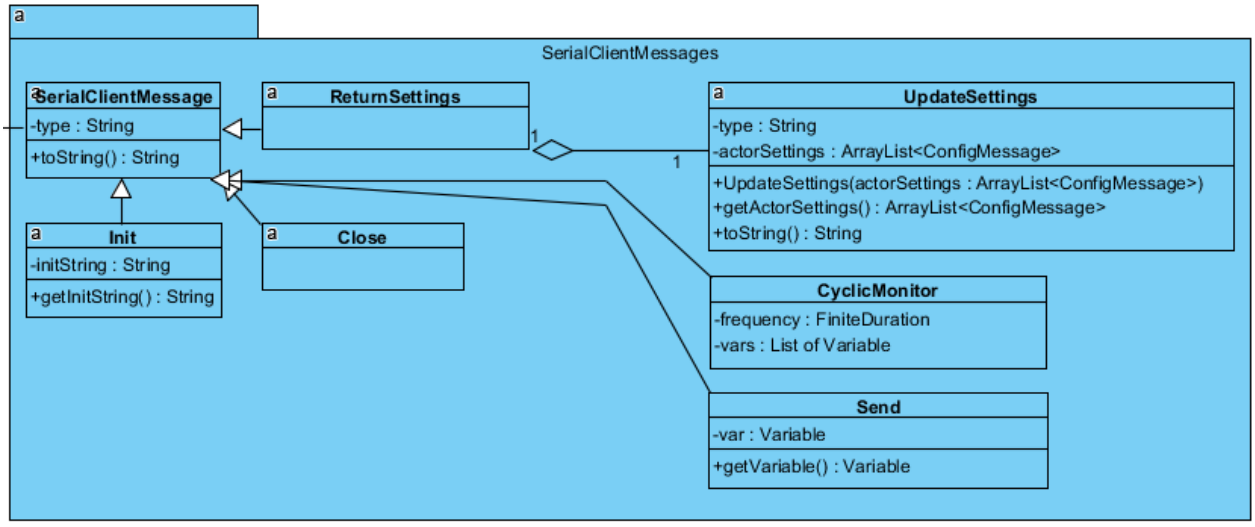
Slouží pro start služby. Modelový aktor odešle do jádra zprávu UseService, ve které je zpráva Init. Pokud není služba v jádru registrována, vytvoří se a předají se jí parametry obsažené ve zprávě. Pokud již existuje, odešle se zpět zpráva ServiceAnswer s informací o tom, že služba je již registrována.

**Send**

Odešle zprávu a čeká na odpověď dobu definovanou při inicializaci. Po obdržení odpovědi nebo vypršení doby čekání na odpověď je odeslána zpráva SerialServiceAnswer do jádra, které ji předá majiteli služby.

**RegisterVar**

Zpráva zaregistruje danou proměnnou do CyclicMonitoru s danou frekvencí.




Obr. 28 Zprávy pro SerialClienta

## 7.5 TCPServer

TCPServer je univerzální předpis vlákna pro komunikaci TCP sokety jako server. Tohoto typu komunikace je využito při komunikaci s addonem v Kodi a s Arduinem.

### 7.5.1 Zprávy pro TCPServer

TCPServer nabízí několik služeb, které mohou uživatelé služeb využívat.

**Init**

Slouží pro start služby. Modelový aktor odešle do jádra zprávu UseService, ve které je zpráva Init. Pokud není služba v jádru registrována, vytvoří se a předají se jí parametry obsažené ve zprávě. Pokud již existuje, odešle se zpět zpráva ServiceAnswer s informací o tom, že služba je již registrována.

**Close**

Ukončení spojení a následně i ukončení činnosti aktora.

**Send**

Odeslání dat typu Variable pomocí TCP portu.

**Received**

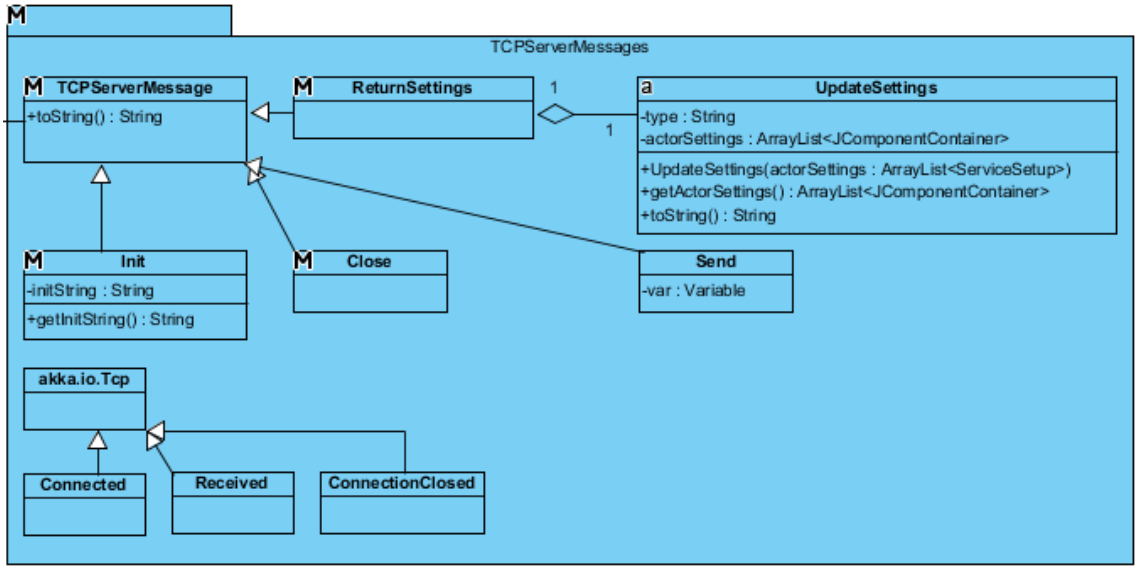
Při přijetí dat na daném portu obdrží aktor, který má port registrovaný zprávu Received. Zpráva v podstatě nahrazuje událost z klasického pojetí komunikace

**Connected**

Při připojení klienta na daný port obdrží aktor zprávu Connected s informací o vytvořeném spojení.

**ConnectionClosed**

Při odpojení klienta z daného portu obdrží aktor zprávu ConnectionClosed.




Obr. 29 Zprávy pro TCPServer

## 7.6 Terminál

Terminál je grafické rozhraní řídicí části aplikace. Je vytvořen pomocí balíku `javax.swing` a pomyslně rozdělen na dvě části. Obě části jsou podrobně popsány v kapitolách 7.6.1 a 7.6.2.

### 7.6.1 Konfigurace

Panel s konfiguracemi slouží ke sledování či nastavení parametrů jednotlivých částí. U každé části systému se v záložkách zobrazují vlastní konfigurace a zároveň i konfigurace využívaných služeb s logem komunikace, jak zobrazuje Obr. 31.

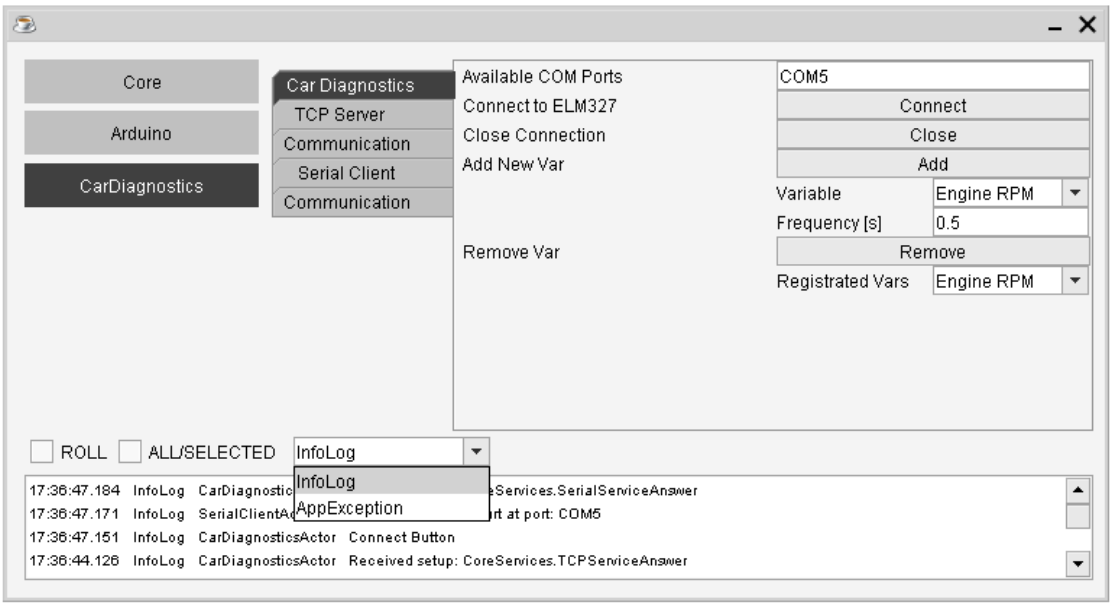
Aktoři posílají své konfigurace do jádra pomocí zprávy UpdateSetup, která zapouzdřuje jinak poměrně složitou datovou strukturu. Jádro pak předá data terminálu. Vyšší seznam definuje záložky, nižší seznam obsahuje řádky konfigurací.

```
public class UpdateSetup extends CoreMessage{

      private ArrayList<ArrayList<JComponentsContainer>> setupList;

      public UpdateSetup(ActorRef senderRef,
      ArrayList<ArrayList<JComponentsContainer>> setupList) {
            super(senderRef);
            this.setupList = setupList;
      }

      public ArrayList<ArrayList<JComponentsContainer>> getSetupList() {
            return setupList;
      }

  }
```

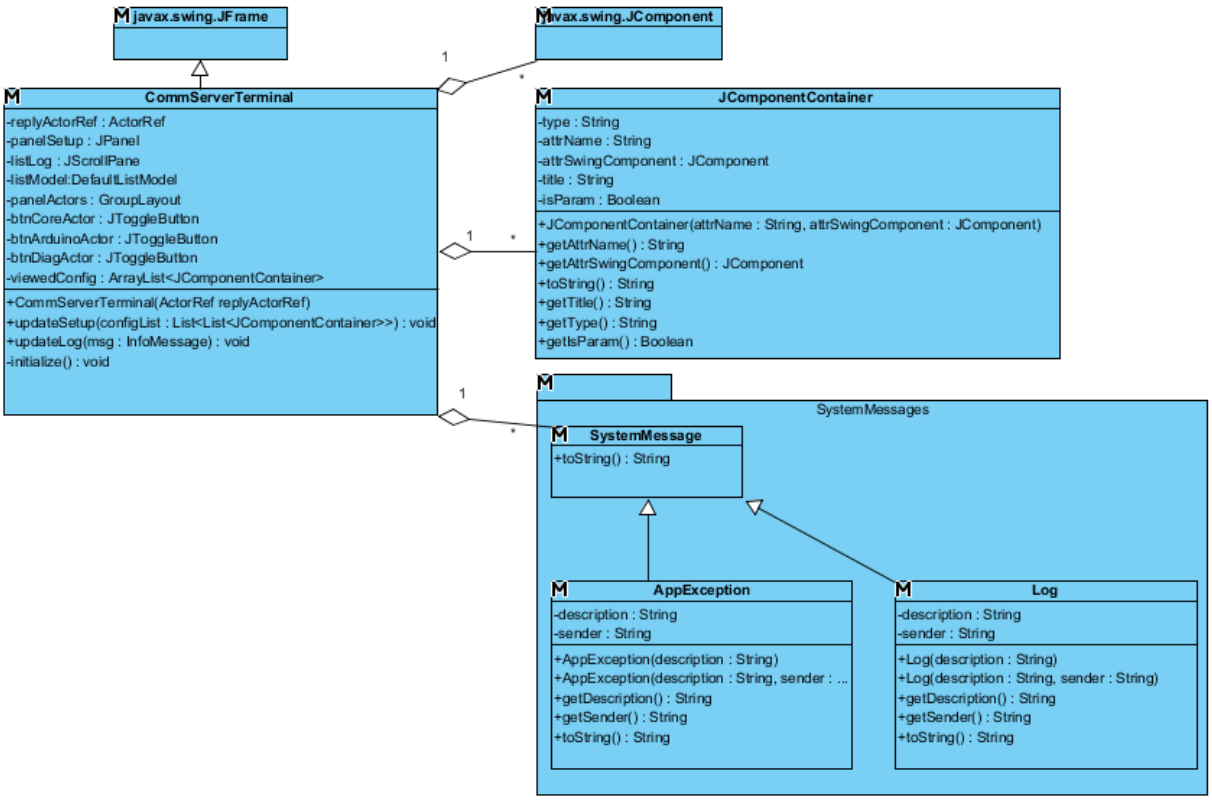
Obr. 30 Třída Update Setup

### 7.6.2 Log

Log zobrazuje buď události z celého systému, nebo jednotlivých částí aplikace. Taktéž lze filtrovat typ zpráv, jak zobrazuje Obr. 31.




Obr. 31 Terminál




Obr. 32 Třídní diagram Terminálu

## 7.7 Arduino Actor

Slouží pro předávání dat mezi addonem a Arduinem. Tato data jsou předávána pomocí TCP protokolu. Pro spojení s addonem slouží port 20000 na adrese localhost, pro spojení s Arduinem port 20000 na adrese 192.168.100.2.

Předávání dat funguje velmi jednoduše. Data přijatá z addonu jsou dešifrována a přeložena do zprávy pro Arduino. Stejným způsobem jsou data zpracována i při opačné cestě.

## 7.8 CarDiagnostics komunikace s ECU

Komunikace s ECU probíhá pomocí převodníku ELM327, který tvoří abstrakci na sběrnicí CAN. Pomocí sériového portu je možné vyčítat data z libovolné dostupné jednotky v automobilu.

### 7.8.1 Knihovna RXTX

Abstrakci nad sériovým portem v Javě tvoří knihovna RXTX, která obsahuje ovladače pro většinu známých operačních systémů. Díky této abstrakci bylo možné vyvíjet SW ve Windows, a poté ho bez úprav nasadit v Raspberry Pi.

### 7.8.2 Funkce zápisu a čtení

Pro komunikaci s ECU byly vytvořeny dvě funkce. Funkce *writeLine(String ... command)* má jako parametr pole textových řetězců, které představují PID příkazy. Funkce *readResponse()* je blokující čekání na odpověď z jednotky. Obě tyto funkce jsou zobrazeny níže.

```
private void writeLine(String... command){
          try {
            for (String commandPart : command) {
              this.port.writeString(commandPart);
            }
            this.port.writeString("\r\n");
          } catch (SerialPortException e) {
            this.coreRef.tell(new AppException(getSelf(),
            e.getMessage()), getSelf());
          }

        }
```

Obr. 33 Funkce writeLine

```
private List<String> readResponse(long timeout){

          try {
            final StringBuilder buffer = new StringBuilder(256);
            final long start = System.currentTimeMillis();
            while (true) {
              if (start + timeout < System.currentTimeMillis()) {
                  this.coreRef.tell(new AppException(getSelf(), "Timeout " +
                  timeout + " occured"), getSelf());
              }

              if (this.port.getInputBufferBytesCount() == 0) {
                continue;
              }
              else{
                  final String string = this.port.readString();
                    buffer.append(string);

                  if (StringUtils.endsWith(string, RESPONSE_TERMINALCHAR))
            {
                     buffer.setLength(buffer.length() - 2);
                     break;
                   }
              }
            }
            final String response = buffer.toString();

            if (response == null || response.trim().isEmpty())
            {
            this.coreRef.tell(new AppException(getSelf(), "Retrieved no
            response from the port"), getSelf());
            }

            final String[] lines = StringUtils.split(response, '\r');
            final List<String> responses = new
            ArrayList<String>(lines.length);
            for (String line : lines) {
              final String trimmed = StringUtils.trimToNull(line);
              if (trimmed != null) {
                responses.add(trimmed);
                System.out.println(trimmed);
              }
            }

            return responses;

          } catch (SerialPortException e) {
            this.coreRef.tell(new AppException(getSelf(), e.getMessage()),
            getSelf());
          }

          return null;

        }
```

Obr. 34 Funkce readResponse

### 7.8.3 Formát PID příkazu

PID příkaz [2] se skládá z definice režimu a poté samotného PID bytu. Příkladem může být příkaz pro vyčtení aktuálních otáček motoru „01 0C“. Většinu obecných PID příkazů uvádí zdroj [2] nebo XML soubor, který je součástí práce.

### 7.8.4 Formát odpovědi od ECU

Odpověď od ECU se skládá z definice příkazu, kterým byla odpověď vyvolána, přičemž k číslu režimu je přičteno číslo 0x40. Následují data dané veličiny.

**Online data**

Každá veličina má definovaný počet bytů v odpovědi a vzorec pro jejich správnou interpretaci. Příkladem mohou být opět otáčky motoru. Odpověď může vypadat např. „41 0C 0C 08“, přičemž vzorec pro dekódování dat pro otáčky motoru je $((A * 256) + B)/4$. Počet bytů odpovědi pro otáčky je 2, což představuje proměnné A a B ve vzorci.

Pro interpretaci dat podle vzorce byla použita knihovna *exp4j*, která zvládne provést tuto interpretaci v průměru za 15 ms, což je poměrně dobrý čas, v porovnání s ostatními knihovnami, kterým tento překlad trvá několikanásobně déle.

**Chybové kódy**

Odpověď pro chybové kódy obsahuje opět hlavičku, která je popsána výše, za níž následují dvojice bytů, definující daný chybový kód, např. „43 03 03 03 03 05 03 06“. Z této odpovědi je možné vyčíst následující informace:

- První byte udává, že se jedná o odpověď na PID příkaz 03, který jednotce řekne, aby vrátila aktuální chybové kódy.
- Druhý byte je počet následujících chybových kódů a vyskytuje se pouze u protokolu 15765-4 (CAN)
- Následující data je nutné rozdělit na dvojice bytů, z nichž každá představuje jeden chybový kód, který lze interpretovat podle Obr. 7. Pro data z příkladu výše by se tedy jednalo o chybové kódy P0303, P0305 a P0306, což jsou chyby spalovacího ústrojí motoru.

**Víceřádková odpověď**

Víceřádková odpověď je použita u dat, které jsou delší než 6 bytů, typicky např. u informací o vozidle v režimu č. 9. Příkladem může být vyčítání VIN kódu, který se skládá až z 38 znaků. Odpověď pro Ford Focus (rv. 2005) je ve tvaru

```
0:490201574630
1:53585847434453
2:38413736333238
```

Interpretace je velmi podobná předchozím dvěma metodám. Jediný rozdíl je v označení čísla řádku, které umožňuje správné seřazení rámců.

## 7.9 CarDiagnostics konfigurace

Konfiguraci diagnostiky automobilů lze rozdělit na dvě části. Definici online proměnných a definici chybových kódů. Seznamy obou typů těchto definic jsou uloženy v XML souborech a načítány při startu diagnostického aktora.

### 7.9.1 Definice online proměnných

Online proměnná je typ dat, který lze zobrazit pomocí grafů. Jedná se především o data v režimu č. 1. Pro tento účel byla vytvořena třída OnlineDataVariable, která obsahuje všechna potřebná data pro vyčítání a správnou interpretaci dané provozní proměnné. Seznam těchto proměnných je získán při spuštění diagnostického aktora z XML souboru. Tvar proměnné je zobrazen níže. Soubor obsahuje přibližně 120 online proměnných připravených ke sledování a logování.

Proměnná obsahuje data pro sestavení příkazu pro ECU a data pro interpretaci odpovědi od ECU. Mimoto také obsahuje limity pro nastavení grafů a jednotku dané veličiny.

```
<CarDiagnosticsComponents.OnlineDataVariable>
  <Name>Engine load</Name>
  <Mode>ONLINE</Mode>
  <PID>04</PID>
  <BytesReturned>1</BytesReturned>
  <Description>Calculated engine load value</Description>
  <MinValue>0.0</MinValue>
  <MaxValue>100.0</MaxValue>
  <Unit>%</Unit>
  <Formula>A*100/255</Formula>
</CarDiagnosticsComponents.OnlineDataVariable>
```

Obr. 35 Definice OnlineDataVariable

### 7.9.2 Definice chybových kódů

Definice chybových kódů jsou uloženy v XML souborech a pro práci s nimi byla vytvořena třída TroubleCodeVariable, jejíž serializovanou podobu lze vidět níže. Celkem lze identifikovat přibližně 5500 chybových hlášek, přičemž tento objem lze rozšířit o kódy jednotlivých výrobců. V práci byl vytvořen XML soubor se specifickými chybovými hláškami pro vozidla Ford.

Proměnná se skládá z definic skupiny, popisu a daného kódu, podle kterého lze proměnnou jednoznačně identifikovat.

```
<CarDiagnosticsComponents.TroubleCodeVariable>
  <Group>Generic</Group>
  <Type>Powertrain - Ignition System and Misfire Detection</Type>
  <Code>P0303</Code>
  <Description>Cylinder 3 Misfire Detected</Description>
</CarDiagnosticsComponents.TroubleCodeVariable>
```

Obr. 36 Definice proměnné TroubleCodeVariable

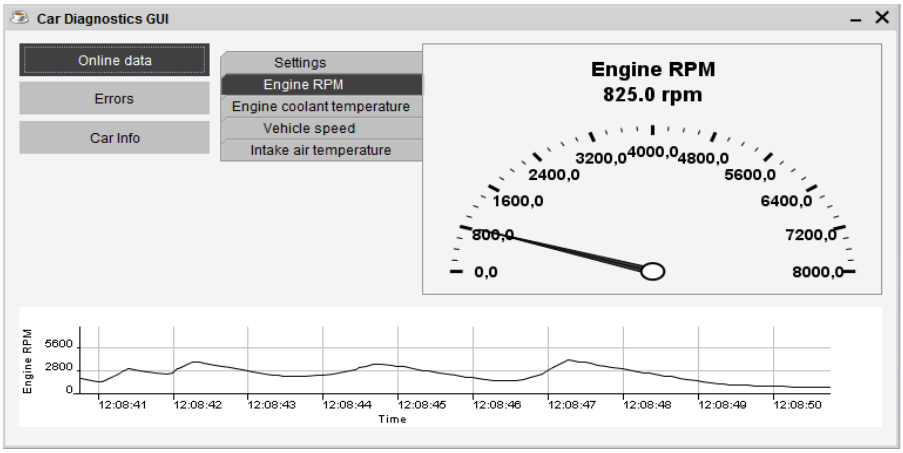
## 7.10 CarDiagnostics vizualizace

Vizualizace využívá dvou grafických komponent. První z nich je úhlový graf DialChart z knihovny JFreeChart [12], který zobrazuje aktuální hodnotu dané veličiny. Druhým je klasický XY graf z knihovny JChart2D [13], který vizualizuje průběh dané veličiny v konstantním časovém intervalu. Oba tyto grafy lze vidět na Obr. 37.

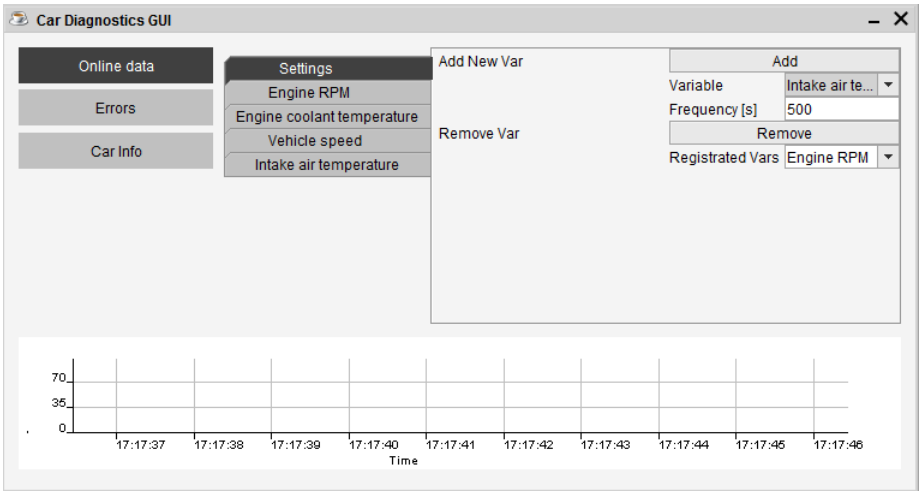
Okno vizualizace se otevírá při vytvoření spojení s jednotkou a rovněž se s ním i zavírá. Je rozděleno na tři části, které popisují následující kapitoly.

### 7.10.1 Online data

V této části diagnostického grafického rozhraní lze sledovat aktuální hodnoty provozních veličin, které se zobrazují pomocí dvou výše zmíněných grafů, které lze vidět na Obr. 37. Program funguje tak, že uživatel registruje vybrané proměnné ke sledování v SerialClient aktoru, který je poté cyklicky odesílá do ECU a odpovědi vrací vlastníkovi služby, tj. v tomto případě diagnostickému aktoru. Registrace a odhlašování proměnných se provádí v záložce „Settings“.



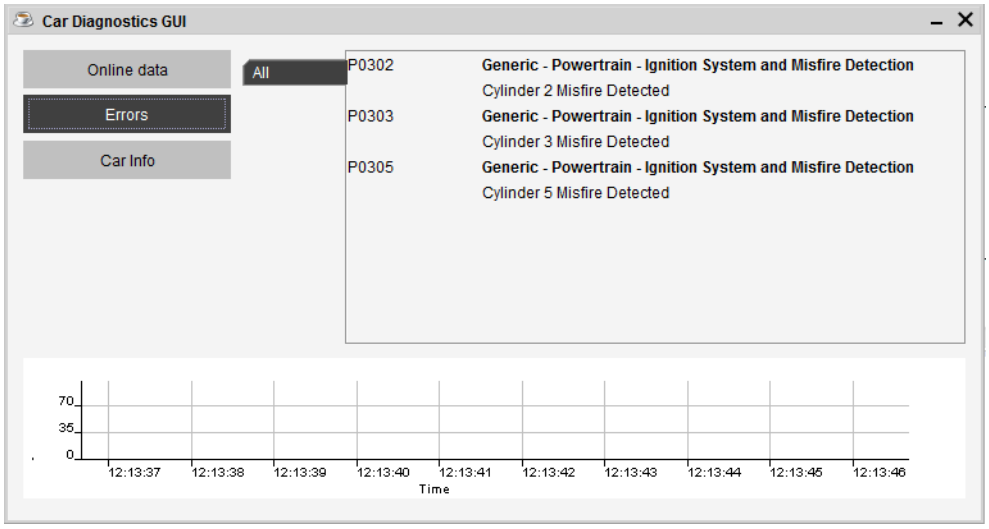

Obr. 37 Vizualizace provozních veličin




Obr. 38 Vizualizace provozních veličin - nastavení

### 7.10.2 Výpis chybových kódů

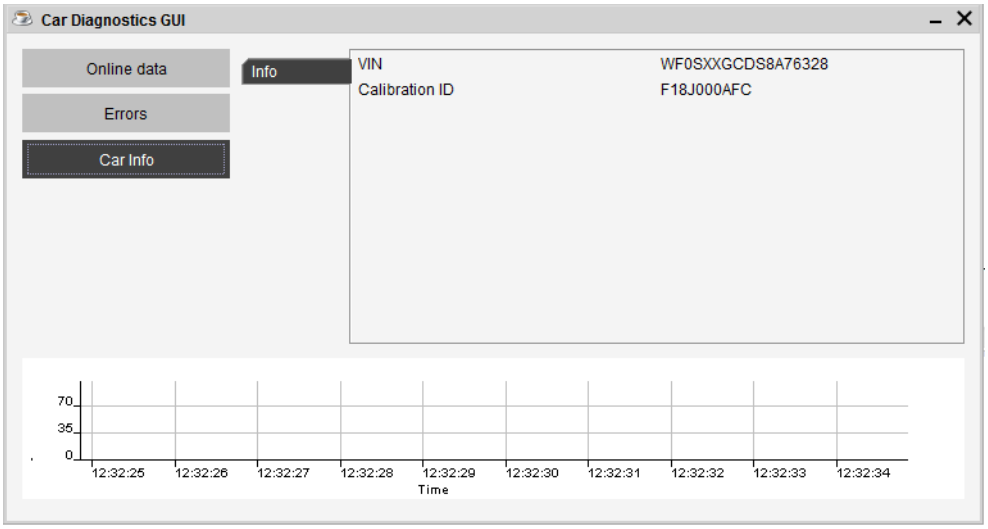
Druhou podstatnou částí diagnostiky je výpis chybových kódů vozidla, který lze vidět na Obr. 39. Pro tento účel byla vytvořena databáze více než 5000 kódů, které lze u každého automobilu rozeznat. Více o definicích těchto kódů pojednává kapitola 7.9.2.




Obr. 39 Výpis chybových kódů

### 7.10.3 Výpis informací o vozidle

Třetí část, zobrazena na Obr. 40, poskytuje základní informace o vozidle jako je VIN kód nebo ID kalibrace.




Obr. 40 Výpis informací o vozidle

## 7.11 FM Radio addon

Addon byl vytvořen pro vizualizaci služby FMRadio, kterou nabízí Arduino. Tento program běží v MMC Kodi a byl vytvořen v jazyce Python [6][7].

### 7.11.1 Knihovna xbmcgui

Knihovna xbmcgui obsahuje tři hlavní třídy. Třída Control obsahuje grafické prvky použitelné pro tvorbu addonu. Rozdíl mezi zbývajícími třídami Dialog a Window je ve způsobu zobrazení addonu. Dialog slouží jen ke sdělení nějaké jednoduché informace, příp. nějaké reakce na ní. Třída Window naopak akceptuje vkládání Controlů, takže lze vytvářet složitější aplikace.

### 7.11.2 Třída Window

Byla vytvořena třída dědící z Window, která inicializuje veškeré grafické prvky. Ke všem prvkům rovněž definuje posluchače a reakce na události.

Třída je volána před připojením k TCP serveru. Podle výsledku pokusu o připojení zobrazuje odlišné prvky, viz obrázky níže.

### 7.11.3 Třída ControlButton

Praktická ukázka vytvoření a zobrazení tlačítka pro přechod na oblíbenou stanici. Dole registrace události nad Controlem, a odeslání zprávy do TCP serveru.

```
import xbmcgui

self.button_next = xbmcgui.ControlButton(1000, 220, 96, 96, '',
os.path.join(dir, 'resources/next.png'),
os.path.join(dir, 'resources/next.png'), 0, 0, 0,
'font13', '0xFFFFFFFF', '0xFFFF3300', 0, '0xFF000000', '0xFF00FFFF')

self.button_next.setEnabled(True)
self.button_next.setVisible(True)

self.addControl(self.button_next)

def onControl(self, controlId):
    if controlId==self.button_next:
        message = 'CMD_next'
        sock.sendall(message)
```

Obr. 41 Ukázka kódu použití tlačítka v Kodi addonu

### 7.11.4 Knihovna socket

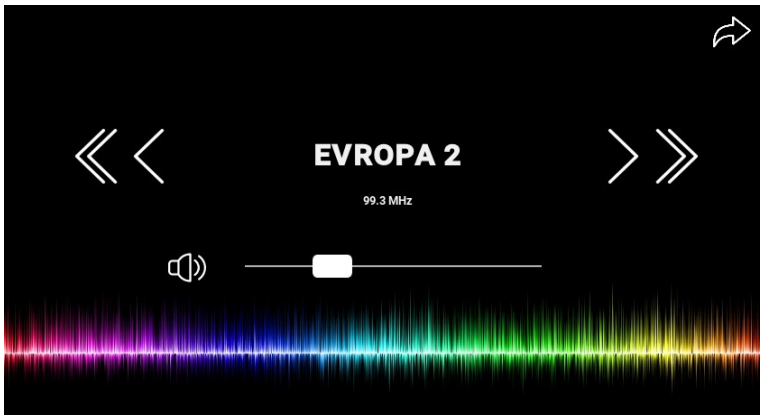
Pro připojení k TCP serveru je nutné vytvořit INET socket. To v Pythonu umožňuje knihovna socket.

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = ('127.0.0.1', 20000)

sock.connect(server_address)
…
…
sock.close()
```

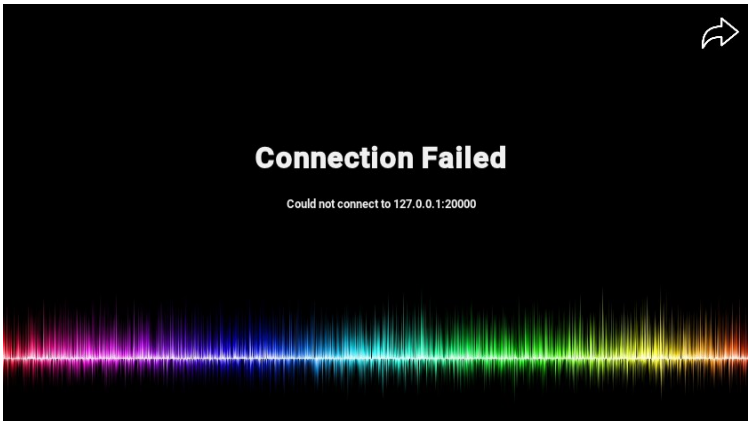
Obr. 42 Ukázka kódu použití knihovny socket

### 7.11.5 Spuštění aplikace

Po spuštění se aplikace pokusí připojit k TCP socketu na IP adrese 127.0.0.1:20000. Pokud se jí to podaří, zobrazí se rozhraní z Obr. 43. Pokud není spuštěn komunikační most, spojení se nepodaří vytvořit a aplikace zobrazí pouze informaci o chybě, viz Obr. 44




Obr. 43 Grafické rozhraní FM addonu




Obr. 44 FM addon chyba spojení

### 7.11.1 Zprávy pro FMRadio addon

Tabulka 9 definuje kódy příchozích zpráv pro FM addon. Tyto zprávy představují příkazy pro Arduino.

Tabulka 9 Odchozí zprávy pro FMRadio addon

| Typ zprávy | Kód zprávy |
|---|---|
| **Seek Up** | 1000 |
| **Seek Down** | 1001 |
| **Next Station** | 1002 |
| **Previous Station** | 1003 |
| **Volume Up** | 1004 |
| **Volume Down** | 1005 |
| **Set Volume** | 1006 |
| **Set Station** | 1007 |

Tabulka 10 definuje kódy odchozích zpráv pro FM addon. Tyto zprávy slouží k aktualizaci grafického rozhraní aplikace.

Tabulka 10 Příchozí zprávy pro FMRadio addon

| Typ zprávy | Kód zprávy |
|---|---|
| **Command OK** | 2000 |
| **Actual Frequency** | 2001 |
| **Actual Volume** | 2002 |
| **Actual Station** | 2003 |
| **Saved Stations** | 2004 |

### 7.11.2 Formát zprávy pro FMRadio addon

Příchozí i odchozí zprávy mají stejný formát:

**[Kód zprávy].[Data]**

Základním prvkem zprávy je identifikátor [Kód zprávy], za kterým mohou následovat data.

# 8 Realizace SW pro Arduino

Realizace SW pro Arduino představuje především vytvoření programu pro ovládání rádia a pro jeho ovládání pomocí TCP protokolu nebo tlačítek na přední liště.

## 8.1 Core

Má na starosti stavové proměnné jednotlivých částí programu a jejich správu. Přijímá data od SerialReader, TCPClient a DigitalWriter. Celé jádro představuje funkce *handle(String cmd),* která implementuje komunikaci mezi částmi systému.

## 8.2 FMRadio

Ovladač FM modulu má na starosti zápis a čtení registrů, které obsahují stavové proměnné procesoru Si4703. Pomocí těchto registrů probíhá komunikace s rádiovým čipem.

Třída *radio.h* umožňuje komunikaci s kterýmkoli zařízením, které obsahuje nějaký FM čip. Třída *si4703.h* je pak od této třídy zděděna. Seznam dostupných funkcí zobrazuje Tabulka 11.

Tabulka 11 Funkce knihovny Si4703.h

| **Funkce** | **Význam** |
|---|---|
| *setVolume(int volume);* | Nastavení hlasitosti (0 až 15) |
| *getVolume()* | Vrací aktuální hlasitost |
| *setMute(bool switchOn)* | Zapnutí potlačení zvuku |
| *getMute()* | Vrací aktuální nastavení potlačení zvuku |
| *setBassBoost(bool switchOn)* | Zapnutí BassBoost módu |
| *getBassBoost()* | Vrací aktuální nastavení BassBoost módu |
| *getMinFrequency()* | Vrací dolní frekvenční limit |
| *getMaxFrequency()* | Vrací horní frekvenční limit |
| *getFrequencyStep* | Vrací minimální frekvenční krok při ladění stanic |
| *setFrequency(RADIO_FREQ newF)* | Nastaví danou frekvenci |
| *getFrequency(void)* | Vrátí aktuální frekvenci |
| *seekUp()* | Ladění směrem nahoru |
| *seekDown()* | Ladění směrem dolů |
| *setMono(bool switchOn)* | Nastaví jednokanálový zvuk |
| *getMono()* | Vrátí nastavení jednokanálového zvuku |
| *checkRDS()* | Vrací příznak přítomnosti RDS |
| *attachReceiveRDS(receiveRDSFunction newFunction)* | Zaregistruje callback funkci pro příjem RDS |

## 8.3 DigitalWriter

Zapouzdřuje operace s digitálními výstupy. Je použit při spínání napájecího relé. Relé je zapojeno jako Normally Open, a při startu je nutné nastavit spínací pin na hodnotu 5 V.

```
void setup() {
//Inicializace a rozepnutí relé
  pinMode(2, OUTPUT);
  digitalWrite(2, HIGH);
}
void loop() {
//Sepnutí relé
If(sepniRele) digitalWrite(2, LOW);
}
```

Obr. 45 Ukázka kódu spínání relé

## 8.4 SerialReader

Sleduje, zda nejsou k dispozici data na USB portu, do kterého je připojena tlačítková lišta čelního panelu. V případě přijetí dat, jsou tato ihned odeslána jádru k vyhodnocení akce.

Tlačítková lišta představuje tzv. HID zařízení, což je standard, definující komunikaci mezi IO zařízeními a počítačem. Pro použití s Arduinem je třeba použít USB Host shield a knihovnu *usb.h*, která obsahuje funkce pro vyčítání reportů ze zařízení, pomocí kterých lze snadno zjistit, které z tlačítek bylo stisknuto.

Použitá tlačítková lišta obsahuje pět tlačítek, které lze použít k libovolným účelům.

Tabulka 12 Akce tlačítek přední levé lišty

| **Kód tlačítka** | **Akce** |
|---|---|
| **0x01** | Sepnutí/rozepnutí relé napájení RPi |
| **0x02** | Zvýšení hlasitosti rádia |
| **0x03** | Snížení hlasitosti rádia |
| **0x04** | Ladění rádia směrem nahoru |
| **0x05** | Ladění rádia směrem dolů |

## 8.5 TCPClient

TCP klient umožňuje komunikaci s Raspberry Pi. Tato komunikace se využívá pro ovládání rádia z Kodi addonu a pro sdělování stavů tlačítek a perfierií vzdálenému systému.

Knihovna Ethernet patří mezi tzv. Arduino knihovny, které usnadňují práci s jeho perfieriemi. Níže popsané třídy jsou využity pro TCP klienta.

**Třída IPAddress**

Třída slouží pro definice všech použitých IP adres. V ukázce kódu lze vidět také definici MAC adresy jako pole bytů.

```
IPAddress ip(192, 168, 100, 2);
IPAddress gateway(192, 168, 100, 1);
IPAddress mask(255, 255, 255, 0);

byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
```

Obr. 46 Ukázka definice parametrů TCP klienta

**Třída EthernetClient**

Třída definuje typ vytvořeného spojení.

Start klienta se spouští funkcí *begin(mac, ip),* která inicializuje TCP klienta. Po skončení této metody je klient připraven připojit se k serveru.

Funkce *connect(ip, port)* umožňuje připojení k serveru. Návratové hodnoty funkce zobrazuje Tabulka 13.

Další důležitou metodou třídy je funkce *available(),* která signalizuje příchozí data.

Ukončení klienta umožňuje funkce *stop()*.

Tabulka 13 Návratové hodnoty připojení k serveru

| **Hodnota** | **Význam** |
|---|---|
| **1** | SUCCES |
| **-1** | TIMED_OUT |
| **-2** | INVALID_SERVER |
| **-3** | TRUNCATED |
| **-4** | INVALID_RESPONSE |

# 9 Testování

Testování probíhá na automobilu Ford Focus, vyrobeného v roce 2005.

Při odesílání cyklických příkazů do jednotky je velmi důležitá odezva. Ta se velmi liší v závislosti na použitém médiu. Testovány byly dva typy přenosu dat. Prvním z nich je bezdrátový Bluetooth, druhým USB.

Jak lze nejspíš tušit, mnohem lepší vlastnosti vykazuje drátová komunikace přes USB rozhraní, která má průměrnou odezvu 60 ms a nízký rozptyl časů (řádově jednotky ms). Při použití bezdrátové komunikace je tato odezva několikanásobně vyšší a i rozptyl časů je poměrně vysoký (řádově desítky ms).

Naproti tomu bezdrátová komunikace má výhodu, že není třeba hledat cestu pro kabeláž od OBD konektoru k USB portu. Přesto byla nakonec zvolena varianta s USB, která má ovšem nevýhodu, že hledání cesty pro kabel se může mírně lišit podle typu vozidla.

# 10 Závěr

Požadavky vytyčené v zadání se podařilo splnit.

Práci jsem řešil způsobem iteračního vývoje softwaru. Každá iterace se skládala z návrhu a realizace dané části práce, přičemž při realizaci se vždy vyskytlo několik problémů, které bylo nutné zohlednit v další iteraci.

Řešení lze rozdělit na dvě části, návrh a realizaci HW a návrh a realizaci SW.

Při realizaci HW části jsem řešil hlavně prostor, neboť bylo nutné umístit poměrně vysoký počet komponent do poměrně malého prostoru, který představuje skříňka na Obr. 22 a Obr. 23. Tento úkol jsem vyřešil skládáním komponent nad sebe, ať už se jedná o shieldy pro Arduino, nebo o použití distančních sloupků. Výsledkem jsou dva sloupce komponent umístěné vedle sebe, takže zbylo dost místa pro kabeláž.

Při realizaci SW části jsem se v počtu iterací dostal k poměrně vysokému číslu, neboť mé původní představy byly mírně idealistické, a jak jsem později zjistil, na Raspberry Pi jen obtížně realizovatelné. Má původní představa se skládala z použití JavaFX a klasického multithreadingu pro vytvoření graficky robustní a výkonné aplikace. Bohužel od JavyFX (varianta FXML) jsem byl nucen odstoupit kvůli velmi nízkému výkonu na Raspberry Pi. Klasický multithreading (třída Thread a rozhraní Runnable) jsem po první iteraci opustil kvůli lepší zkušenosti s Actor frameworkem. V další iteraci jsem již použil javax.Swing a Akka Framework, se kterými jsem pak pracoval až do konce, nicméně ještě byly nutné dvě iterace pro vyladění architektury aktorů a grafického rozhraní diagnostiky.

Za vlastní přínos dané problematice považuji především použití actor frameworku na Raspberry Pi. Actor Framework Akka má nespornou výhodu virtuálního stroje Javy, kdy lze odladit aplikaci na jakémkoliv OS, který podporuje Javu a poté program již zkompilovaný přenést na Raspberry Pi. Za další přínos považuji realizaci HW, neboť se jedná o modulární systém, který lze kdykoliv bez větších zásahů rozšířit nebo modifikovat podle vlastních potřeb. Také cena celého modulu se pohybuje okolo 6000 Kč, což je v porovnání s komerčními produkty poměrně dobrý kompromis, vzhledem k tomu že je k dispozici plnohodnotný OS s řadou open source produktů a diagnostika automobilu.

Z hlediska budoucího vývoje je třeba zvážit možné alternativy k Raspberry Pi 3. Jako poměrně dobrá alternativa se jeví jednodeskové počítače Banana Pi, které lze na rozdíl od Raspberry Pi rozšířit o HDD připojeným pomocí SATA rozhraní. Další možnou alternativou se zdá být jednodeskový počítač řady Mini ATX, který disponuje procesory x86 a x64 a lze na nich tedy provozovat běžné OS.

## Seznam použité literatury


[1] *ISO-15031: E/E Diagnostic Test Modes*. 1. Washington D.C.: Výkonné nakladatelství federálního registru, 2002.

[2] PIDs. *Escape PHEV TechInfo* [online]. Seattle: SEVA, 2007 [cit. 2017-04-16]. Dostupné z: http://www.eaa-phev.org/wiki/Escape_PHEV_TechInfo#PIDs

[3] HORÁK, B., K. FRIEDRISCHKOVÁ, J. KAZÁRIK, J. NOWAKOVÁ a Z. SLANINA. [i]Elektromobilita II, učební text.[/i] 1. vyd. Ostrava: VŠB-TU Ostrava, 2014. ISBN 978-80-248-3532-7. Dostupné z: http://netfei.vsb.cz/downloads/autorske_texty/Elektromobilita%20II.pdf.

[4] MCCORD, Keith. [i]Automotive Diagnostic Systems: Understanding OBD-I & OBD-II (S-A Design Workbench Series).[/i] CarTech, 2011. ISBN 978-1934709061.

[5] *Official Kodi Wiki* [online]. Australia: Lifehacker, 2014 [cit. 2017-04-24]. Dostupné z: http://kodi.wiki/

[6] *Python.org* [online]. U.K.: Python Software Foundation, 2001 [cit. 2017-04-24]. Dostupné z: https://www.python.org/

[7] MONK, Simon. [i]Programming the Raspberry Pi, Second Edition: Getting Started with Python.[/i] 2 ed., McGraw-Hill Education, 2015. ISBN 978-1259587405.

[8] *Raspberry Pi* [online]. U.K.: Raspberry Pi Foundation, 2015 [cit. 2017-04-24]. Dostupné z: https://www.raspberrypi.org/

[9] *Raspbian* [online]. U.K.: Raspberry Pi Foundation, 2015 [cit. 2017-04-24]. Dostupné z: https://www.raspbian.org/

[10] KISZKA, Bogdan. *1001 tipů a triků pro jazyk Java*. 1. Praha: COMPUTER PRESS, 2012, 544 s. ISBN 9788025124673.

[11] *Akka* [online]. California: Lightbend, 2014 [cit. 2017-04-24]. Dostupné z: http://akka.io/

[12] *JFree* [online]. U.K.: Object Refinery Limited, 2005 [cit. 2017-04-24]. Dostupné z: http://www.jfree.org/

[13] JChart2D. *SourceForge* [online]. U.K.: Slashdot Media, 2017 [cit. 2017-04-24]. Dostupné z: http://jchart2d.sourceforge.net/

[14] Princip a použití Lambda sondy. *Automatizace.HW.cz* [online]. ČR: HW server, 2014 [cit. 2017-04-24]. Dostupné z: http://automatizace.hw.cz/view.php%3Fcisloclanku%3D2006061301

[15] Catalyst - AECC. *AECC* [online]. Belgium: AECC, 2015 [cit. 2017-04-24]. Dostupné z: http://www.aecc.eu/technology/catalysts/

# Seznam příloh

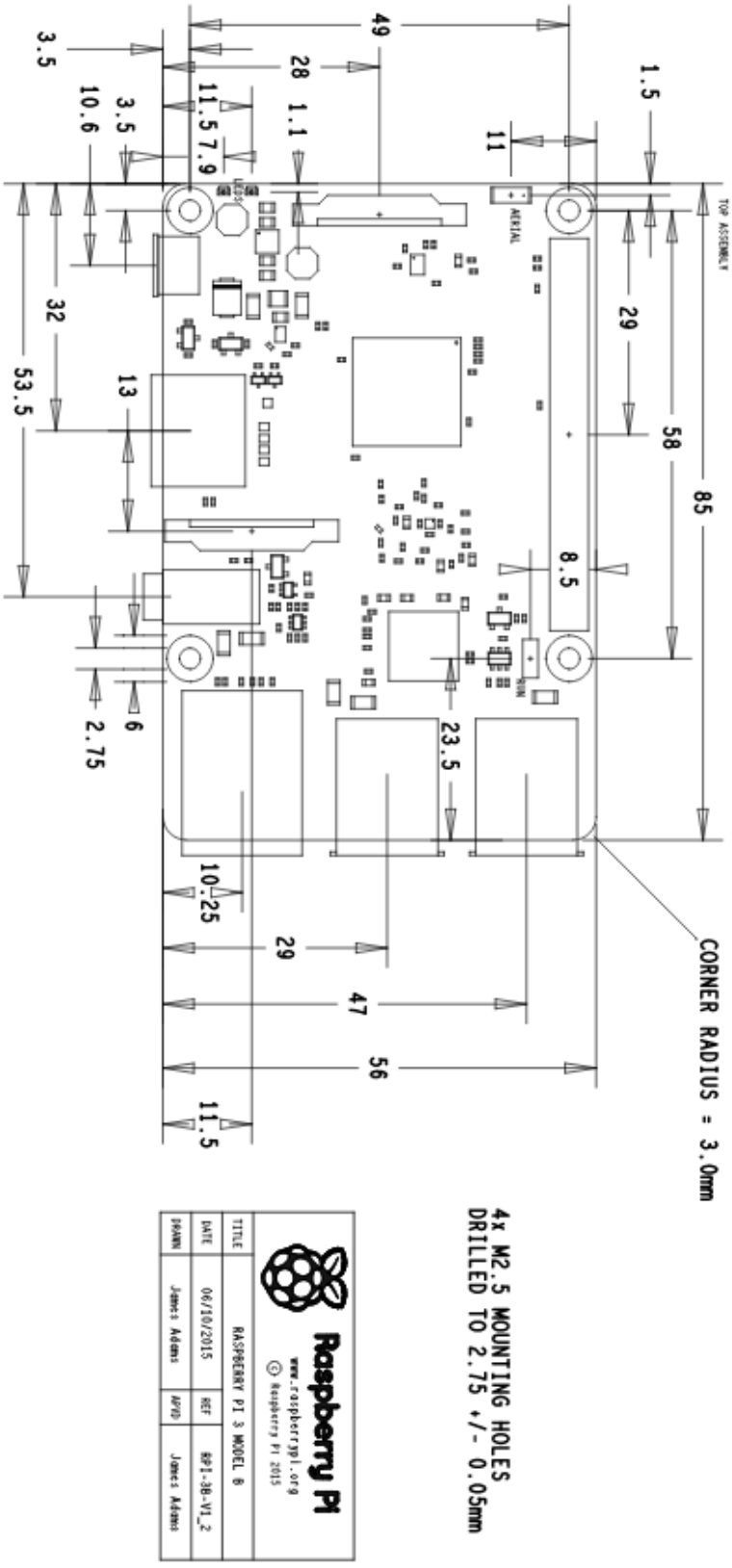
# Příloha A

Seznam doplňků použitých pro realizaci HW platformy

| **Doplněk** | **Počet** |
|---|---|
| Cuprextit 200x160x1,5 jednovrstvý | 1 ks |
| Nepájivé kontaktní pole ZY-170 W | 1 ks |
| Drátové propojky | 10 ks |
| Svorka bezšroubová WAGO222-415 | 3 ks |
| Distanční sloupek DA5M3X05S nikl | 4 ks |
| Distanční sloupek DA5M2X10 | 4 ks |
| Distanční sloupek DA5M2X30 | 4 ks |
| Šroub M3X10 | 6 ks |
| Šroub M2X10 | 4 ks |
| Matice M3 | 4 ks |
| Matice M2 | 4 ks |
| Konektor Jack 3,5 mm Male | 2 ks |
| Kabel HDMI Male/Male 20 cm | 1 ks |
| Kabel USB Male/Male 20 cm | 3 ks |
| Kabel MicroUSB Male 20 cm | 1 ks |

# Příloha B

Schéma desky Raspberry Pi 3



# Příloha C

Použitý procesor pro komunikaci s ECU

## ELM327

## OBD to RS232 Interpreter

### Description

Almost all of the automobiles produced today are required, by law, to provide an interface for the connection of diagnostic test equipment. The data transfer on these interfaces follow several standards, but none of them are directly usable by PCs or smart devices. The ELM327 is designed to act as a bridge between these On-Board Diagnostics (OBD) ports and a standard RS232 serial interface.

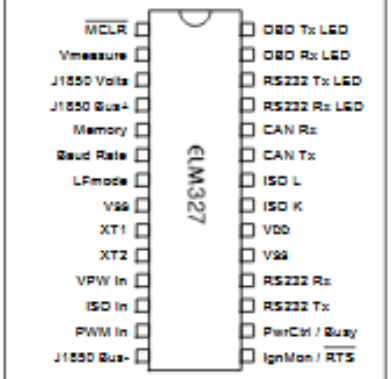
In addition to being able to automatically detect and interpret nine OBD protocols, the ELM327 also provides support for high speed communications, a low power sleep mode, and the J1939 truck and bus standard. It is also completely customizable, should you wish to alter it to more closely suit your needs.

The following pages discuss all of the ELM327's features in detail, how to use it and configure it, as well as providing some background information on the protocols that are supported. There are also schematic diagrams and tips to help you to interface to microprocessors, construct a basic scan tool, and to use the low power mode.

### Features

- Power Control with standby mode
- Universal serial (RS232) interface
- Automatically searches for protocols
- Fully configurable with AT commands
- Low power CMOS design

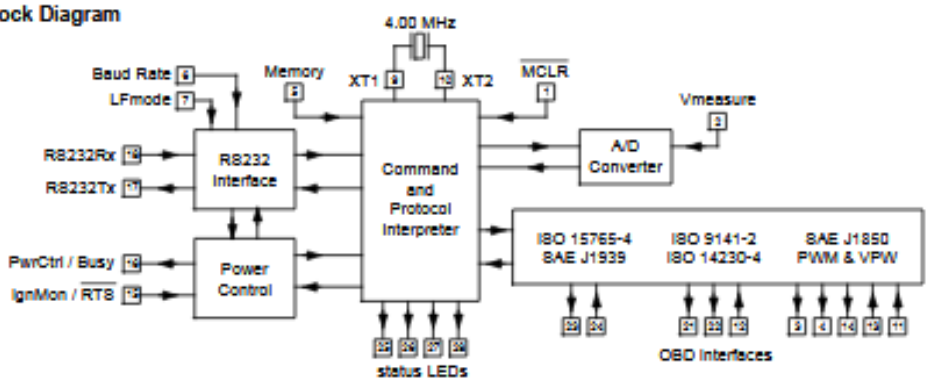
Connection Diagram
PDIP and SOIC
(top view)




### Applications

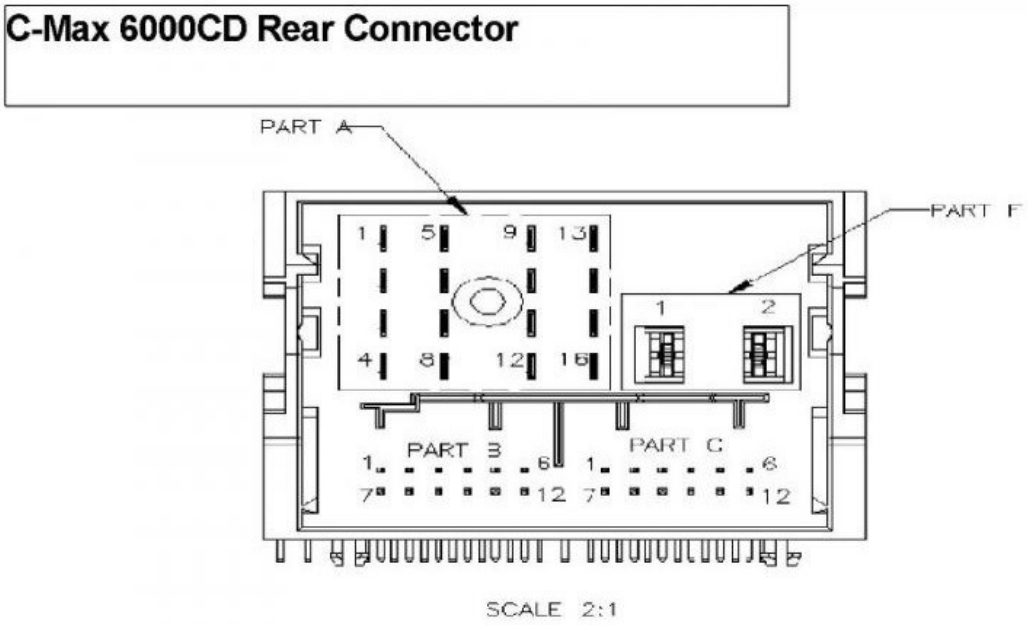
- Diagnostic trouble code readers
- Automotive scan tools
- Teaching aids

### Block Diagram



# Příloha D

Schéma konektorů původního autorádia




Connector A

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Pin 1 | Right Rear + | Pin 5 | Right Rear - | Pin 9 | CAN + | Pin 13 | Alarm Sense |
| Pin 2 | Right Front + | Pin 6 | Right Front - | Pin 10 | CAN – | Pin 14 | Illumination |
| Pin 3 | Left Front + | Pin 7 | Left Front - | Pin 11 | Gound | Pin 15 | Battery (+) |
| Pin 4 | Left Rear + | Pin 8 | Left Rear - | Pin 12 | Masse | Pin 16 | Ignition |

| Connector B | | Connector C | | Connector C | | | |
|---|---|---|---|---|---|---|---|
| Pin 1 | Mono + | Pin 7 | Mono – | Pin 1 | RSE L + | Pin 7 | RSE L – |
| Pin 2 | Geschalteter Plus | Pin 8 | SWC Gnd | Pin 2 | RSE R + | Pin 8 | RSE R – |
| Pin 3 | Aux L + | Pin 9 | Aux L – | Pin 3 | Mono 2 + | Pin 9 | Mono 2 – |
| Pin 4 | Aux R + | Pin 10 | Aux R – | Pin 4 | Sub W + | Pin 10 | Sub W – |
| Pin 5 | Telefon Mute | Pin 11 | Reverse Sense | Pin 5 | Aux 2 L | Pin 11 | Aux 2 R |
| Pin 6 | SWC | Pin 12 | (AVC) | Pin 6 | Audio Gnd | Pin 12 | Mic. |

# Příloha E

Adresářová struktura přiloženého CD

| Složka | Popis obsahu |
|---|---|
| **/DP** | Diplomová práce v PDF formátu |
| **/Datasheets** | Obsahuje datasheety použitých součástek |
| **/SW** | Obsahuje veškerý vyvinutý SW |
| **/SW/Arduino** | Obsahuje zdrojové kódy pro Arduino |
| **/SW/RaspberryPi/Java** | Obsahuje zdrojové kódy řídicí aplikace |
| **/SW/RaspberryPi/Kodi** | Obsahuje zdrojové kódy FM addonu |